LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
Department "Institut für Informatik"
Lehr- und Forschungseinheit Medieninformatik
Prof. Dr. Heinrich Hußmann

Diplomarbeit

Semi-automatische Accessibility-Analyse von Webseiten

Andreas Singer
singer@cip.ifi.lmu.de

Bearbeitungszeitraum:	01.06.2006 bis 30.11.2006
Betreuer:	Dipl.-Inf. Richard Atterer
Verantwortlicher Hochschullehrer:	Prof. Dr. Heinrich Hußmann

Kurzzusammenfassung

In dem stark wachsenden World Wide Web ist Accessibility ein wichtiges Thema. Immer mehr Firmen und Organisationen bieten dort Services an, die rund um die Uhr und von überall auf der Welt benutzt werden können. Gerade für Menschen mit Einschränkungen bringt das enorme Vorteile. Umso wichtiger ist es, dass man einen barrierefreien und gleichberechtigten Zugang zu den Inhalten im Web sicherstellt. Aus diesem Grund wurden vom World Wide Web Consortium, aber auch von nationalen Gesetzgebern, umfangreiche Richtlinienkataloge erstellt, an die man sich beim Verfassen von Inhalten für das Web halten soll.

Um Webautoren bei der Umsetzung dieser Regeln zu unterstützen, wurden automatische Accessibility-Evaluatoren entwickelt, die Verstöße gegen die Richtlinien feststellen und bei deren Behebung helfen sollen. Leider ist der Automatisierungsgrad aktueller Evaluatoren noch nicht weit fortgeschritten, weshalb die Prüfung einer Webseite sehr zeitintensiv ist und ohne Expertenwissen oder umfangreiche Kenntnis der Regelwerke nicht bewältigt werden kann.

Diese Arbeit soll deshalb Bereiche identifizieren, in denen die Effektivität solcher Programme verbessert werden kann. Dazu werden zuerst aktuelle Entwicklungen im Bereich der Accessibility-Regelwerke aufgezeigt. Danach betrachten wir vorhandene Accessibility-Tools, um uns einen Überblick über deren Schwachstellen zu verschaffen und um Bereiche zu identifizieren, wo Verbesserungen möglich sind. Das darauf folgende Kapitel führt in das eng mit der Accessibility verknüpfte Themengebiet der Readability ein. Es werden Verfahren zur strukturellen und inhaltlichen Analyse von Texten vorgestellt und nach Möglichkeiten gesucht, diese in vorhandene Evaluatoren zu integrieren. Auch die Farbgebung des Textes und daraus resultierende Probleme sollen dabei betrachtet werden.

Im 5. Kapitel wird der Prototyp eines semi-automatischen Accessibility-Evaluators erweitert und es werden einige der zuvor identifizierten Verbesserungsmöglichkeiten eingebaut und deren Umsetzung erläutert. Das Kapitel „Implementierung“ beschäftigt sich dann detailliert mit der Programmierung der Hauptpunkte der neuen Verfahren. Am Ende dieser Ausarbeitung werden die umgesetzten Lösungen evaluiert und mit anderen gängigen Tools verglichen. Die Arbeit schließt mit der Zusammenfassung der Ergebnisse und einem Ausblick auf zukünftige Ansätze und Weiterentwicklungen auf diesem Gebiet.

Abstract

The World Wide Web is growing fast and Accessibility is a major concern. Nowadays many companies, organisations and authorities offer their services online and as a consequence customers or visitors can get access at any time and from any place in the world. This can be an enormous advantage for disabled persons and so it is very important to develop Web content that is accessible for everyone. For this reason the World Wide Web Consortium has released a set of guidelines which should be concerned when developing for the Web.

Additionally, software for automatic Accessibility testing has been developed in order to help Web authors to fulfil the requested checkpoints and to alert them if something goes wrong. Unfortunately current Accessibility checkers lack of tests that can be carried out automatically. Manual checking is very time-consuming and needs expertise or a deep knowledge of the guidelines.

This work shall find possibilities to increase the effectiveness of those tools. Therefore we take a look at the available Accessibility guidelines first. Afterwards an overview of current Accessibility tools and their limitations is presented. Based on the findings there, we try to find areas where these tools can be improved. The next section introduces methods and tools for Readability checking and tries to find ways to integrate them into a Web Accessibility evaluator.

In chapter 5 a prototype of a semi-automatic Accessibility checker will be extended with functionality to support our improvements. Details of the implementation will be given in the next chapter. At the end of this work, the implemented methods will be evaluated and the prototype will be compared to other Accessibility software. Finally all the findings of this work will be summarized and an outlook to future work and improvements in this sector is given.

Aufgabenstellung

für die Diplomarbeit in Informatik:

Bearbeiter: **Andreas Singer**
Immatrikulations-Nr.: **18003517**

Thema: Semi-automatische Accessibility-Analyse von Webseiten

Ziel der Diplomarbeit ist die Erstellung bzw. die Erweiterung eines Prototyps, welcher Webdesigner bei der Einhaltung und Umsetzung von Usability- und Accessibility-Richtlinien unterstützt. Dabei soll im Gegensatz zu vielen bereits vorhandenen Werkzeugen vor allem auch zusätzlich auf den Kontext bzw. die Bedeutung (Metadaten) der Inhalte der betrachteten Webseite geachtet werden, wodurch viele Richtlinien zuverlässiger und besser beurteilt werden können. Die Arbeit wird dabei folgende Aufgaben umfassen:

- Überblick über Guidelines/Regelwerke aus dem Bereich der Web-Accessibilty
- Analyse und Beurteilung von bereits vorhandenen, bekannten Accessibility-Tools
- Überblick über Theorien und Richtlinien in den Bereichen der Wortschatzanalyse und der Readability (Textstruktur, Farben/Kontraste)
- Implementierung/Erweiterung eines Tools mit Möglichkeiten zur Klassifizierung des Bildcontents (Inhalt oder Layout) und zur Festlegung der Zielgruppe sowie darauf aufbauenden automatisierten Tests von de facto Accessibility-Standards
- Analyse des Seitenmodells mit/ohne CSS (bzw. Scripten) und Beurteilung der Verwendbarkeit laut den Vorgaben der Regelwerke
- Bewertung der Lesbarkeit der Seite anhand der Textmerkmale/Kontraste

In der Analysephase soll außerdem die Architektur des zu erweiternden Tools betrachtet werden, die im Systementwurf unter Umständen für die Erweiterungen entsprechend angepasst werden muss.

Verantwortlicher Hochschullehrer: **Prof. Dr. Heinrich Hußmann**
Betreuer: **Dipl.-Inf. Richard Atterer**

Beginn am: **01.06.2006**
Einzureichen am: **30.11.2006**

Selbstständigkeitserklärung

„Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig angefertigt, alle Zitate als solche kenntlich gemacht sowie alle benutzten Quellen und Hilfsmittel angegeben habe."

München, den 30. November 2006

_____
Andreas Singer

Inhaltsverzeichnis

Abbildungs- und Tabellenverzeichnis

Abbildungen

Tabellen

Auflistungs- und Formelverzeichnis

Auflistungen

Formeln

1 Einführung

1.1 Motivation

Das rasante Wachstum des Internets in den letzten Jahren hat dazu geführt, dass in Industrieländern wie den USA, oder vielen mitteleuropäischen Staaten wie Deutschland bereits bis zu 60 % aller Haushalte über einen Computer mit Internetzugang verfügen. Bei Unternehmen liegt diese Quote gar annähernd bei 90 % [37]. Diese große Reichweite hat aber auch eine Explosion der Inhalte und Services, die im Internet angeboten werden, mit sich gebracht. Firmen, die heutzutage nicht im World Wide Web (WWW) vertreten sind, laufen Gefahr, ein wichtiges Marktsegment unbearbeitet zu lassen und langfristig Kunden zu verlieren. Das Geschäft im Internet ist hart, denn nirgendwo sonst ist die Konkurrenz so orts- und zeitunabhängig „nah" wie im weltweiten digitalen Netz. Ein Klick genügt, um den Anbieter zu wechseln und die Angebote unterscheiden sich oft nur noch durch ihre Präsentation oder durch die Art, wie schnell und einfach ein potentieller Kunde an sein gewünschtes Ziel kommen kann [13].

Aber nicht nur der E-Commerce macht sich das Internet zu nutze, immer häufiger sind auch öffentliche Institutionen „online" und bringen ihre Services und Informationen so nah wie möglich zum Bürger. E-Government, die „elektronische Verwaltung", ist das Schlagwort für Dinge wie den Steuerausgleich am eigenen PC oder den elektronischen Krankenschein. Der Verwaltungsapparat spart sich dadurch Kosten und die Bürger kommen in den Genuss von quasi unbegrenzten „Öffnungszeiten" und bequemer Dateneinsicht von zu Hause aus [29].

Insgesamt gab es 2005 geschätzte 9 Milliarden Seiten im Web, die von Suchmaschinen indiziert waren. Die Angabe einer genauen Zahl ist natürlich unmöglich, da das Internet hochdynamisch ist und sich laufend ändert [14]. Aber allein die Größenordnung dieser Schätzung zeigt, in welchen Dimensionen man sich hier bewegt. Diese enorme Größe, die steigende Anzahl an öffentlich relevanten Services und die Möglichkeit, dass jeder Inhalte im Internet publizieren kann, machen es natürlich nötig, dass man verstärkt darauf achtet, dass diese Inhalte allen Menschen gleichermaßen zur Verfügung stehen und zugänglich sind. Das World Wide Web Consortium versucht mit der Schaffung von Standards und Richtlinien Ordnung in das Web zu bringen, um dessen Zugänglichkeit sicherzustellen.

Doch auch die besten Ratschläge von Experten nutzen wenig, wenn der gewöhnliche Webautor sie nicht überblicken kann oder selbst ein Experte sein muss, um sie umzusetzen. Ganz zu schweigen von dem enormen Zeitaufwand, den die vollständige Prüfung einer Webseite mit sich bringt. Deshalb wird versucht, Werkzeuge zu entwickeln, die bereits während der Erstellung von Inhalten, Hinweise auf mögliche Problemfelder geben und bei deren Behebung helfen, um standardkonforme Seiten zu erhalten. Weiters gibt es eine Fülle von Programmen, die fertig gestellte Webseiten scannen und analysieren und den Autoren Verletzungen der Richtlinien und mögliche Zugänglichkeitsbarrieren aufzeigen.

Leider scheitert der effektive Einsatz solcher Werkzeuge oft daran, dass zu wenige Kontextinformationen verfügbar sind, um bestimmte Kriterien zuverlässig zu beurteilen. So ist es für ein Programm durchaus ein Problem festzustellen, wo sich auf einer Webseite Bereiche wie die

Navigation oder der Hauptinhaltsbereich befinden, oder ob sie überhaupt vorhanden sind. Es ist nicht einfach, zu beurteilen, ob auf der Seite dargestellte Informationen passend für die gewünschte Zielgruppe formuliert wurden, oder ob ein Alternativtext oder eine Beschreibung auch zu dem entsprechenden Bild passt oder gar nicht nötig ist. Viele dieser Grauzonen werden von heute im Einsatz befindlichen Evaluatoren einfach zur manuellen Überprüfung markiert oder teilweise falsch bewertet und unnötigerweise als Fehler ausgegeben. Damit wird ein guter Teil der zeitlichen Einsparungen, die eine automatische Evaluierung bringen soll, wieder zunichte gemacht, da diese manuelle Nacharbeit erst Recht wieder zeitintensiv ist und eventuell erneut Expertenwissen benötigt [2].

Im Sinne des Semantic Web [56] benötigt man also Metainformationen, um die Bedeutung von Inhalten auch für Maschinen verwertbar zu machen, damit diese sinngemäß und effizient darauf reagieren können.

1.2 Ziele

Das Ziel dieser Arbeit besteht darin, die Effektivität von derzeitigen Accessibility-Evaluatoren zu verbessern. Die Ergebnisse von automatischen Auswertungen sollen genauer werden und weniger Fehltreffer beinhalten. Außerdem soll versucht werden, manuelle Checkpunkte zu automatisieren und den Prüfprozess dadurch zu beschleunigen. Im Rahmen der Arbeit müssen daher Bereiche identifiziert werden, wo derartige Verbesserungen möglich sind. Darauf aufbauend soll ein Prototyp erstellt beziehungsweise erweitert werden, der die Umsetzung einiger ausgewählter Punkte mit Verbesserungspotential demonstriert.

1.3 Struktur

Die Arbeit beginnt in Kapitel 2 mit der Vorstellung von aktuellen Accessibility-Regelwerken. Dabei werden zuerst generelle Empfehlungen des Word Wide Web Consortium und danach nationale Umsetzungen davon betrachtet. Es werden Unterschiede und Probleme herausgearbeitet und diskutiert.

Das dritte Kapitel bietet eine Übersicht über bekannte, kostenlose Software zur automatischen Evaluierung von Accessibility-Kriterien. Die Schwachstellen und Unzulänglichkeiten der Evaluatoren werden identifiziert, um daraus Potentiale für zukünftige Prüfprogramme abzuleiten.

Im darauf folgenden, vierten Kapitel geht es um das Thema Readability. Nach einer kurzen Einleitung werden Formeln vorgestellt, die versuchen die Lesbarkeit eines Textes über strukturelle Merkmale zu berechnen. Danach werden Möglichkeiten erörtert, mit denen man das Vokabular beziehungsweise den Wortschatz von Texten analysieren kann. Weiters geht es noch um den Einfluss der Farbwahl auf Kontrastverhältnisse und Lesbarkeit. Am Ende des Kapitels werden dann Readability-Tools betrachtet, die versuchen, einige der zuvor genannten Punkte umzusetzen. Abschließend werden die Ergebnisse des Kapitels zusammengefasst und diskutiert.

Kapitel 5 greift die Erkenntnisse der vorangegangenen Abschnitte auf und versucht daraus Verbesserungsmöglichkeiten abzuleiten, die sich zur Umsetzung eignen. Die Umsetzung dieser Bereiche basiert auf einem Prototyp, dessen grundsätzliche Architektur und Funktionsweise vorgestellt wird. Diese Architektur wird dann entsprechend erweitert, um die Anforderungen dieser Arbeit erfüllen zu können. Das Kapitel schließt mit einer Funktionsbeschreibung des neuen, erweiterten Prototypen.

Im Kapitel „Implementierung“ werden anschließend die Hauptbereiche des neuen Evaluators detailliert beschrieben. Dabei geht es vor allem um die Umsetzung neuartiger oder verbesserter Analyseverfahren. Es werden Methoden zur verbesserten Analyse von Alternativtexten, Farb-

kontrasten und strukturellen Eigenschaften von Webseiten sowie Verfahren zur Beurteilung der Lesbarkeit vorgestellt und implementiert.

Diese Neuerungen werden danach im 7. Abschnitt dieser Arbeit evaluiert und mit den Ergebnissen anderer Accessibility-Tools verglichen. Zum Schluss werden noch einmal alle wichtigen Punkte kurz zusammengefasst und ein Ausblick für zukünftige Arbeiten gegeben.

1.4 Zentrale Begriffe

1.4.1 Usability

Die Usability, oder Gebrauchstauglichkeit, ist ein Qualitätsmerkmal und bezieht sich darauf wie einfach oder intuitiv ein Interface für einen Benutzer zu bedienen ist. Jakob Nielsen, einer der führenden Experten im Usability-Bereich [28], gibt fünf Komponenten an, die zur Gebrauchstauglichkeit beitragen:

1. Erlernbarkeit
2. Effizienz
3. Merkbarkeit
4. Fehlerbehandlung, Fehlertoleranz
5. Zufriedenstellung

Laut Nielsen besagt das erste Gesetz des E-Commerce, dass ein potentieller Kunde ein Produkt nur kaufen kann, wenn er es auch findet. Und die erste Aktion, die ein Kunde im Web ausführt, wenn er auf Schwierigkeiten trifft oder etwas nicht finden oder bedienen kann, ist die Webseite zu verlassen und sich eine bessere zu suchen. Eine die mehr auf seine Bedürfnisse eingeht und über eine bessere Usability verfügt [27].

1.4.2 Accessibility

Unter Accessibility (Zugänglichkeit oder Barrierefreiheit) kann man ein Teilgebiet oder ein angrenzendes Gebiet der Usability verstehen. Hier wird versucht Benutzerschnittstellen so zu gestalten, dass sie allen möglichen Anwendern, unabhängig von körperlichen oder technischen Einschränkungen (Plattformunabhängigkeit), gleichermaßen zur Verfügung stehen. Man sieht sofort, dass die Begriffe Usability und Accessibility deshalb eng verknüpft sind, denn ein Interface, welches aus bestimmten Gründen für Benutzer kaum zu erlernen, effizient zu bedienen oder tolerant gegenüber Eingabefehlern (z. B. von Menschen mit Koordinationsproblemen) ist, kann auch keine gute Zugänglichkeit aufweisen [18].

1.4.3 Readability

Der letzte zentrale Begriff dieser Arbeit beschäftigt sich schließlich mit der Lesbarkeit, der Readability, von Content (Inhalten). Readability wird auch als die Accessibility von Schriftstücken gesehen, denn etwas, das von Benutzern (oder Maschinen) nicht erfasst, verstanden und weiterverarbeitet werden kann, kann auch nicht zugänglich sein.

2 Accessibility-Regelwerke

In diesem Kapitel werden bekannte Accessibility-Standards eingeführt und beschrieben. Dabei wird zuerst auf aktuelle und zukünftige Empfehlungen des World Wide Web Konsortiums eingegangen und danach werden einige nationale Umsetzungen davon vorgestellt und die Unterschiede hervorgehoben. Abschließend wird die Sinnhaftigkeit der Regelwerke kritisch betrachtet.

Das Statistische Bundesamt Deutschland bezifferte für Dezember 2003 die Anzahl der in Deutschland lebenden Menschen mit staatlich anerkannten Behinderungen mit ca. 8,5 Millionen. 6,6 Millionen davon waren sogar als schwer behindert eingestuft. Diese Zahlen allein bestätigen, dass Accessibility ein wichtiges Thema ist, da es sich hierbei bereits um ca. 10 % der Gesamtbevölkerung handelt und die Berücksichtigung von leichteren Einschränkungen wohl noch deutlich höhere Zahlen liefern würde. Zudem steigt der Altersschnitt unserer Gesellschaft beständig an und erwartungsgemäß treten auch Behinderungen gehäuft bei Personen im fortgeschrittenen Alter auf (ca. 75 % aller Fälle bei Personen über 55 Jahren). Gerade bei Menschen mit Einschränkungen ist eine Nutzung des Internets (80 %) aber relativ häufig. Dies ist sicherlich auch darauf zurückzuführen, dass das Web bisher noch nie da gewesene, orts- und zeitunabhängige Zugangs- und Interaktionsmöglichkeiten zu Informationen und Angeboten bietet. Zum Beispiel können Personen mit körperlichen Beeinträchtigungen viele Dinge bequem von zu Hause aus über den Computer erledigen. Vor allem in vielen öffentlich relevanten Bereichen wie in der Bildung, im Gesundheitswesen oder in Anwendungen des E-Governments, aber auch in der Wirtschaft, im Arbeitswesen und bei Freizeitangeboten findet das Web immer mehr Verbreitung. Deswegen bedarf es Regelungen und Gesetze, die dafür sorgen, dass allen Bevölkerungsgruppen ein barrierefreier Zugang zu diesen Inhalten ermöglicht wird [38][17].

2.1 Allgemeines

Das World Wide Web Consortium (W3C) ist eine Organisation, die es sich zum Ziel gesetzt hat, den Fortschritt des Webs voranzutreiben und dabei seine Interoperabilität und Zugänglichkeit sicherzustellen. Das W3C wurde im Jahre 1994 gegründet (unter anderem auch von Tim Berners-Lee, dem Erfinder des World Wide Web) und entwickelt seither Spezifikationen, Richtlinien und Tools, die zur Erreichung und Einhaltung dieser Web-Interoperabilität beitragen und eine Fragmentierung des Webs verhindern sollen [50].

Eine zentrale Rolle spielt dabei die Web Accessibility Initiative (WAI), die Bemühungen anstellt, ein barrierefreies (bzw. barrierearmes) Web zu erreichen. Das bedeutet, dass der Zugang und die Nutzung von Webseiten, möglichst unabhängig von den körperlichen oder technischen Möglichkeiten der Nutzer, gewährleistet werden soll. Dabei sollen einerseits Behinderungen aller Art, aber auch ganz natürliche, altersbedingte Einschränkungen (z. B. ein Nachlassen der Sehkraft) und nicht zuletzt technische Barrieren (z. B. ältere, kleinere Bildschirme oder verschiedene Browser) berücksichtigt werden [60]. Um dieses Ziel zu erreichen, gibt die WAI Empfehlungen und Richtlinien für Technologien der Entwicklerseite (Authoring Tool Accessibility Guidelines, ATAG [51]), für Technologien der Benutzerseite (User Agent Accessibility

Guidelines, UAAG [59]) und für den Content selbst heraus (Web Content Accessibility Guidelines, WCAG). Abbildung 2.1 unten zeigt eine Übersicht der drei Bereiche, auf die mittels Accessibility-Richtlinien Einfluss genommen werden soll.

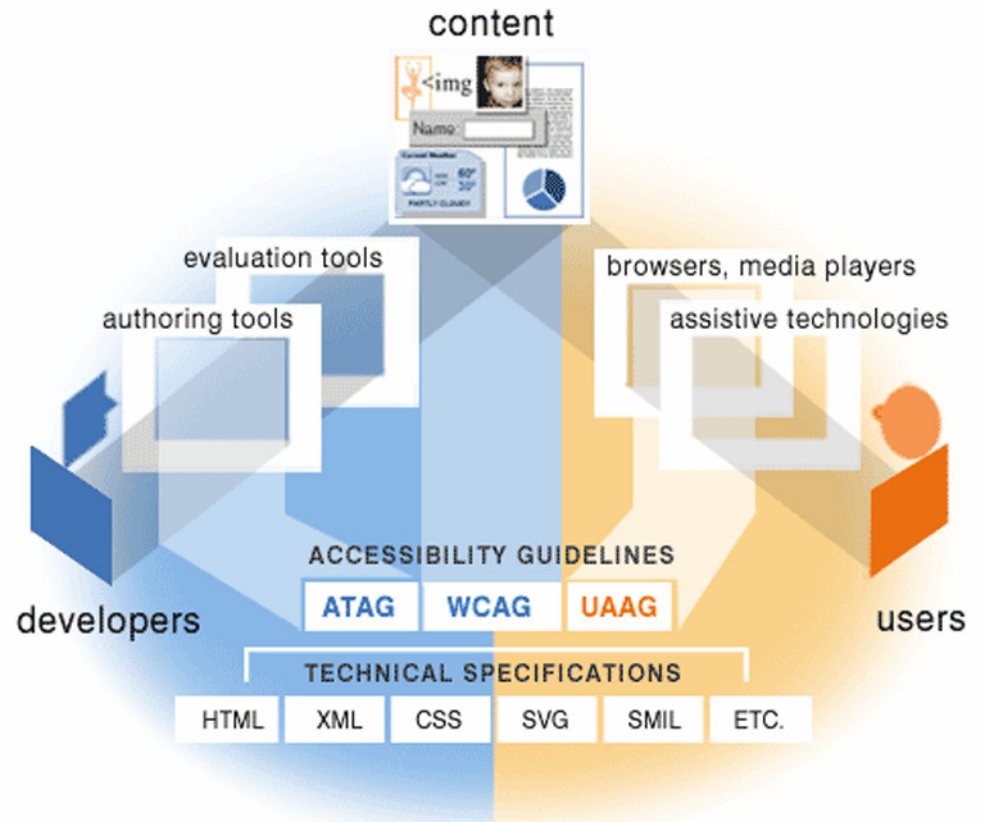


Abbildung 2.1: Bereiche der Web Accessibility Initiative [53]

2.2 Web Content Accessibility Guidelines 1.0

Die Web Content Accessibility Guidelines 1.0 (WCAG) sind eine W3C Empfehlung aus dem Jahre 1999. Sie stellen die derzeit noch aktuelle und gültige Version dar und beinhalten Richtlinien gleichermaßen für Entwickler von Webseiten und Autorenwerkzeugen. Die Richtlinien verfolgen den Zweck, Webinhalte auch Personen mit Einschränkungen zugänglich zu machen, sie beziehen sich aber auch auf eine höhere/bessere Verfügbarkeit des Contents für alle „normalen“ Benutzer. Dies betrifft dann vor allem technologische Einschränkungen des Endgeräts, des Browsers oder der Umgebung, die zur Interaktion benutzt werden. Während der Entwicklung von Webinhalten muss beachtet werden, dass Benutzer möglicherweise

- nicht alle oder gar keine Informationen hören, sehen oder verarbeiten können,
- Schwierigkeiten haben, Text zu lesen oder zu verstehen,
- körperlich nicht in der Lage sind eine Tastatur oder eine Maus zu verwenden,
- keinen graphischen Browser oder ein sehr kleines Display haben,
- die Sprache der Webseite nicht verstehen,
- in der gegenwärtigen Umgebung ihre Augen, Ohren, Hände, … nicht uneingeschränkt benutzen können (z. B. wegen Umgebungslärm),
- eine veraltete Browserversion oder ein anderes Betriebssystem haben,
- oder eine langsame Internetverbindung haben [61].

Solche Probleme stehen in einem heterogenen, verteilten System wie dem World Wide Web an der Tagesordnung und mit Hilfe der Zugänglichkeitsrichtlinien des W3C sollen Webinhalte schon während ihrer Entwicklung robust genug für diese Umstände gemacht werden beziehungsweise im Nachhinein standardisiert geprüft und verbessert werden können.

2.2.1 Prioritätsklassen und Konformitätslevel

Für die einzelnen 14 Richtlinien wurden Checkpunkte (in Summe 65 Stück) ausgearbeitet, die von zugänglichen Webseiten erfüllt werden sollen und deren Einhaltung in drei Prioritätsklassen eingeteilt ist:

- Priorität 1 muss erfüllt werden, ansonsten ist es bestimmten Gruppen nicht möglich, auf die Information zuzugreifen.
- Priorität 2 soll erfüllt werden, da sich sonst der Zugriff bestimmter Gruppen auf die Information schwierig gestaltet.
- Priorität 3 kann umgesetzt werden, um den Zugriff zum Webdokument weiter zu erleichtern.

Passend dazu drückt ein Konformitätslevel A die Erfüllung aller Priorität 1-Checkpunkte aus, Level Double-A erfüllt alle Checkpunkte der Priorität 1 und 2 und Level Triple-A schließlich erfüllt alle drei Prioritätsklassen komplett [61].

2.2.2 Richtlinien und Checkpunkte

Richtlinie 1
„Stellen Sie äquivalente Alternativen für Audio- und visuellen Inhalt bereit."

Hierbei geht es vor allem um die Bereitstellung von Text-Äquivalenten für Nicht-Text-Inhalte wie Bild-, Audio- oder Videodaten, da diese Äquivalente für Menschen mit Einschränkungen besser zugänglich gemacht werden können. So ist es zum Beispiel kein Problem, einen Text mittels Sprachgenerator „vorlesen" zu lassen oder ihn auf speziellen Blindenschrift-Displays auszugeben. Doch auch der umgekehrte Fall (Nicht-Text-Äquivalente wie Bilder oder Tonaufzeichnungen für Textpassagen) kann für einige Anwender nützlich sein, wenn man nur an Menschen mit Leseschwächen denkt.

Checkpunkte zur Erfüllung von Richtlinie 1 (auszugsweise/exemplarisch):

- ✓ Alternativtexte und Beschreibungen für Bilder, Videos, Audios, Frames, Scripts, … zum Beispiel mittels der „alt" und „longdesc" Attribute in HTML (Priorität 1)
- ✓ redundante Textlinks für serverseitige Imagemaps (Priorität 1)
- ✓ Synchronisation von Alternativen (Untertitel, Audiokommentare) zu zeitgesteuerten Multimedia-Inhalten (z. B. Filmen) (Priorität 1)

Richtlinie 2
„Verlassen Sie sich nicht auf Farbe allein."

Menschen mit Farbsehschwächen können Informationen, die sich rein aus deren Farbcodierung (oder der Sichtbarkeit des Kontrastwertes) ergeben, möglicherweise nicht wahrnehmen und auch technologische Hindernisse wie Schwarz/Weiß-Monitore müssen dabei berücksichtigt werden.

Checkpunkte zur Erfüllung von Richtlinie 2 (auszugsweise/exemplarisch):

- ✓ alternative Darstellung von farbcodierten Informationen (Priorität 1)
- ✓ ausreichender Kontrast zwischen Vorder- und Hintergrund (Priorität 2)

Richtlinie 3
„Verwenden Sie Markup und Stylesheets auf korrekte Weise."

Markup-Elemente müssen entsprechend ihrer Grammatik verwendet werden, obwohl manche Browser möglicherweise auch fehlerhaftes oder unvollständiges Markup korrekt interpretieren. Assistive Technologien können viel exakter arbeiten, wenn das Markup entsprechend seiner Spezifikation verwendet wurde. Dies betrifft auch eine klare Trennung des strukturellen Teils der Webseite von den Präsentationseffekten (Layout/Formatierung).

Checkpunkte zur Erfüllung von Richtlinie 3 (auszugsweise/exemplarisch):

- ✓ Validierung des Dokuments gegen veröffentlichte, formale Grammatiken (Priorität 2)
- ✓ wenn möglich, Verwendung von Markup anstelle von Bildinformationen (z. B. für mathematische Formeln) (Priorität 2)
- ✓ Kapselung der Layoutinformationen in Stylesheets (Priorität 2)
- ✓ Bevorzugung von relativen Einheiten in Attributwerten (Priorität 2)

Richtlinie 4
„Verdeutlichen Sie die Verwendung natürlicher Sprache."

Abkürzungen und Akronyme können von assistiven Technologien möglicherweise nicht korrekt interpretiert/angezeigt werden, wenn sie nicht als solche kenntlich gemacht werden bzw. im Dokument keine ausgeschriebene Version davon vorhanden ist. Ebenso müssen die Sprache bzw. Sprachänderungen des Dokuments im Markup angegeben werden, um beispielsweise Sprachgeneratoren einen fließenden Wechsel zu ermöglichen.

Checkpunkte zur Erfüllung von Richtlinie 4 (auszugsweise/exemplarisch):

- ✓ Verwendung des „lang"-Attributs für Änderungen der Sprache des Dokumenttexts (Priorität 1)
- ✓ Angabe der ausgeschriebenen Version von Abkürzungen im „title"-Attribut der entsprechenden Elemente (Priorität 3)

Richtlinie 5
„Erstellen Sie Tabellen, die geschmeidig transformieren."

Tabellen sollten wirklich nur für die Darstellung von tabellarischen Daten verwendet werden und nicht zu Layoutzwecken.

Checkpunkte zur Erfüllung von Richtlinie 5 (auszugsweise/exemplarisch):

- ✓ Kennzeichnung von Zeilen- und Spaltenüberschriften (Priorität 1)
- ✓ Positionierung von Elementen mittels Stylesheets und nicht mit Hilfe von Tabellen (Priorität 2)

- ✓ Angabe von Zusammenfassungen für den Tabelleninhalt (Priorität 3)

Richtlinie 6
„Sorgen Sie dafür, dass Seiten mit neuer Technologie geschmeidig transformieren."

Bei der Verwendung von neuen Technologien wie CSS oder Scripts muss ein Entwickler immer bedenken, dass eine Seite auch dann noch entsprechend sinnvoll angezeigt werden sollte, wenn diese Zusatztechnologien bei machen Nutzern nicht vorhanden oder deaktiviert sind.

Checkpunkte zur Erfüllung von Richtlinie 6 (auszugsweise/exemplarisch):

- ✓ Sicherstellung der Lesbarkeit eines Dokuments auch ohne Stylesheets (Priorität 1)
- ✓ Funktionalität der Seite nicht von Scripts abhängig machen oder geeignete Alternativen (serverseitig) anbieten (Priorität 1)

Richtlinie 7
„Sorgen Sie für eine Kontrolle des Benutzers über zeitgesteuerte Änderungen des Inhalts."

Manche Menschen (z. B. mit kognitiven, visuellen oder körperlichen Einschränkungen) können dynamische Inhalte nicht schnell genug erfassen oder bedienen, weshalb der Benutzer Möglichkeiten haben sollte, die Geschwindigkeit solcher Animationen zu kontrollieren oder diese komplett zu stoppen.

Checkpunkte zur Erfüllung von Richtlinie 7 (auszugsweise/exemplarisch):

- ✓ Vermeidung von Bildschirmflackern (Priorität 1)
- ✓ kein Blinken, keine Bewegung oder periodische Refreshes (Priorität 2)

Richtlinie 8
„Sorgen Sie für direkte Zugänglichkeit eingebetteter Benutzerschnittstellen."

Wenn ein Objekt mit einer eigenen Schnittstelle eingebettet wird, muss dieses – wie die eigentliche Webseite auch – einen geräteunabhängigen Zugriff auf seine Funktionalität gewährleisten.

Checkpunkte zur Erfüllung von Richtlinie 8 (auszugsweise/exemplarisch):

- ✓ Kompatibilität von programmierten Elementen mit assistiven Technologien sicherstellen (Priorität 1)

Richtlinie 9
„Wählen Sie ein geräteunabhängiges Design."

Eine Webseite soll so konzipiert werden, dass der Benutzer mit dem Eingabegerät seiner Wahl darauf zugreifen kann. Elemente, die nur mit einer bestimmten Art von Eingabegeräten benutzt werden können (z. B. Zeigegeräte), schließen möglicherweise bestimmte Benutzergruppen (z. B. Blinde) aus.

Checkpunkte zur Erfüllung von Richtlinie 9 (auszugsweise/exemplarisch):

- ✓ Imagemaps auf Clientseite statt auf Serverseite (Priorität 1)
- ✓ logische vor geräteabhängigen Eventhandlern bevorzugen (Priorität 2)

Richtlinie 10
„Verwenden Sie Interim-Lösungen."

Auch ältere Browser oder assistive Technologien sollen bestmöglich unterstützt werden, zumindest solange bis die Benutzeragenten die Behandlung der folgenden Checkpunkte selbst übernehmen können.

Checkpunkte zur Erfüllung von Richtlinie 10 (auszugsweise/exemplarisch):

- ✓ keine Verwendung von Popup-Fenstern (Priorität 2)
- ✓ exakte/eindeutige Positionierung von Beschriftungen zu Formularfeldern (Priorität 2)
- ✓ lineare Textalternativen für mehrspaltige Textlayouts mit Tabellen (Priorität 3)

Richtlinie 11
„Verwenden Sie W3C-Technologien und –Richtlinien."

Das W3C entwickelt seine Technologien immer im Hinblick auf Zugänglichkeit und prüft diese auch frühzeitig im Entwicklungsprozess, um optimale Ergebnisse sicherzustellen. Außerdem werden W3C-Technologien offen und nicht proprietär entwickelt, um eine breitere Palette von Hard- und Software zu unterstützen und damit eine größtmögliche Zugänglichkeit für alle Menschen zu erreichen.

Checkpunkte zur Erfüllung von Richtlinie 11 (auszugsweise/exemplarisch):

- ✓ überholte Features mit den aktuellen ersetzen (z. B. font-Element vs. „font"-Attribut in CSS) (Priorität 2)
- ✓ Bereitstellung einer kompletten, alternativen Seite, wenn die Originalseite aus bestimmten Gründen auch nach besten Bemühungen nicht zugänglich gemacht werden kann (Priorität 1)

Richtlinie 12
„Stellen Sie Informationen zum Kontext und zur Orientierung bereit."

Unter diesen Punkt fallen zum Beispiel die Gruppierung von zusammengehörenden Elementen oder deren sinnvolle Benennung, um Rückschlüsse auf Inhalt und Zweck zu ermöglichen.

Checkpunkte zur Erfüllung von Richtlinie 12 (auszugsweise/exemplarisch):

- ✓ Frames müssen eine Bezeichnung über das „title"-Attribut erhalten (Priorität 1)
- ✓ Formularelemente werden beispielsweise mit FIELDSET und OPTGROUP gruppiert (Priorität 2)

Richtlinie 13
„Stellen Sie klare Navigationsmechanismen bereit."

Unter diese Richtlinie fallen Checkpunkte, die dafür sorgen, dass ein Besucher einer Webseite jederzeit weiß, wo er sich auf der Seite gerade befindet und wie er (schnellstmöglich) an sein gewünschtes Ziel kommen kann. Dabei ist es auch sehr wichtig, solche Mechanismen einheitlich zu verwenden, um Wiedererkennungseffekte oder transformiertes Anwenderwissen aus anderen Applikationen verwertbar zu machen.

Checkpunkte zur Erfüllung von Richtlinie 13 (auszugsweise/exemplarisch):

- ✓ aussagekräftige Linktexte anstelle von „hier klicken" (Priorität 2)
- ✓ Erstellung einer Sitemap (Inhaltsverzeichnis) (Priorität 2)
- ✓ konsistente Verwendung der Navigationsmechanismen (Priorität 2)
- ✓ unterscheidungskräftige/aussagekräftige Informationen am Beginn von Überschriften, Absätzen, ... um Benutzern, die seriell auf die Seite zugreifen, schnellstmöglich Entscheidungshilfen zu bieten (Front-Loading) (Priorität 3)

Richtlinie 14
„Sorgen Sie dafür, dass Dokumente klar und einfach gehalten sind."

Eine leichte und klare Verständlichkeit der Seite ist nicht nur wichtig für Besucher mit kognitiven Einschränkungen, sondern nimmt großen Einfluss auf das subjektive Zufriedenheitsgefühl eines Benutzers und entscheidet damit maßgeblich über Erfolg oder Misserfolg eines Internetauftritts.

Checkpunkte zur Erfüllung von Richtlinie 14 (auszugsweise/exemplarisch):

- ✓ Verwendung einer klaren, einfachen und angemessenen Sprache (Priorität 1)
- ✓ Einbau von audiovisuellen Hilfsmitteln, wo es das Seitenverständnis fördert (Priorität 3) [61]

Um die vorgestellten Richtlinien zu überprüfen, empfiehlt das W3C sowohl den Einsatz automatischer Tools, als auch eine manuelle Überprüfung durch Menschen, wo eine automatische Überprüfung nicht mehr möglich ist. Begonnen werden soll damit natürlich bereits in den frühesten Entwicklungsstadien, um Probleme bereits im Ansatz zu erkennen und Änderungskosten gering zu halten.

Auf gängige, automatische Accessibility-Evaluatoren wird in Kapitel 3 dieser Arbeit eingegangen.

2.3 Web Content Accessibility Guidelines 2.0

Der technische Fortschritt sowie Probleme und Einschränkungen, die bei der praktischen Anwendung der WCAG 1.0 auftraten, machten es schon bald nötig, das Regelwerk zu überarbeiten und an aktuelle Anforderungen anzupassen und mit der Entwicklung einer Nachfolgerversion zu beginnen.

2.3.1 Ziele der Version 2.0

Seit dem Release der WCAG 1.0 Mitte 1999 sammelte die WCAG Arbeitsgruppe Feedback über die Benutzbarkeit der oben beschriebenen Guidelines und ließ diese Informationen in die Anforderungen und die Weiterentwicklung des Regelwerkes zur Version 2.0 einfließen. Dabei wurden sechs zusätzliche Ziele identifiziert und in die Anforderungen der WCAG 2.0 aufgenommen:

1. Eine technologieunabhängige Anwendbarkeit des Regelwerks soll sichergestellt werden. Wurde in der Version 1.0 noch hauptsächlich für HTML-Dokumente geschrieben, so soll eine generischere Formulierung dafür sorgen, dass die Regeln zukünftig auf mehr als nur eine Markup-Sprache (bzw. ein Dokumentformat) angewendet werden können (CSS, SML, XML, …).
2. Die Konformitätsanforderungen müssen noch klarer formuliert werden und deren Erfüllung muss für den Anwender überprüfbar sein. Zu diesem Zweck sollen mehr und genauere Erfolgskriterien (Checkpunkte), Evaluierungstechniken und Beispiele angegeben werden.
3. Es soll für eine klare und einfache Struktur der Regeln gesorgt werden, damit die Benutzer sie leichter und schneller erfassen, verstehen und besser im Kopf behalten können.
4. Die WCAG 2.0 sollen für eine breitere Leserschaft ausgelegt werden (Webdesigner, Entwickler von Autorenwerkzeugen, Entwickler von Evaluatoren, …) und es soll eine klare und einfache Sprache verwendet werden, die einfach in andere Sprachen übersetzt werden kann.
5. Genauere Angaben über die Vorteile der Regeln und über Personengruppen, die von zugänglichen Inhalten profitieren, werden in das Dokument eingebaut.
6. Da bereits viele Tools, Spezifikationen und andere Organisationen auf die WCAG 1.0 Bezug nehmen, sollen neue Versionen „rückwärtskompatibel“ sein, gleichzeitig aber so offen entwickelt werden, dass sie auch mit zukünftigen Technologien bestmöglich arbeiten können („vorwärtskompatibel“) [54].

Der finale Entwurf (Last Call Working Draft) der Web Content Accessibility Guidelines 2.0 vom 27. April 2006 stellt aus Sicht der WCAG Arbeitsgruppe eine stabile Version dar, die sich aller offenen Punkte früherer Entwürfe angenommen hat. Das Erscheinen der W3C Empfehlung wird noch Ende 2006 oder gleich zu Beginn von 2007 erwartet [62].

2.3.2 Die vier Prinzipien

Das neue Regelwerk enthält in seiner momentanen Fassung 13 Richtlinien, die sich um vier zentrale Prinzipien gruppieren:

Prinzip 1
„Inhalte müssen wahrnehmbar sein.“

Unter diesen Punkt fallen Richtlinien für das Anbieten von Textalternativen für sämtliche Nicht-Text-Inhalte sowie synchronisierte Alternativen für Multimedia-Inhalte. Weiters verlangt das Prinzip 1 eine klare Trennung der Struktur des Dokuments von dessen Layout/Formatierung und es fordert eine einfache Unterscheidbarkeit von Vordergrund- und Hintergrundinformationen.

Hier sind unter anderem inhaltlich die Richtlinien 1, 2 und 3 der WCAG 1.0 wiederzufinden.

Prinzip 2
„Interfacekomponenten innerhalb des Contents müssen bedienbar/zugänglich sein."

Hier geht es darum, sämtliche Funktionalität der Seite per Tastatur zugänglich zu machen. Außerdem muss den Benutzern die Möglichkeit gegeben werden, zeitliche Abläufe der Interaktion zu kontrollieren, sowie Bereiche zu vermeiden oder auszuschalten, die zu Problemen bei Personen mit erhöhter Photosensitivität führen können (Blinken, Flackern, …). Schließlich verlangt die Erfüllung von Prinzip 2 noch Mechanismen zur Suche, Orientierung und Navigation im Content und eine hohe Fehlertoleranz gegenüber Benutzereingaben und –aktionen.

Hier sind unter anderem inhaltlich die Richtlinien 7, 8, 9, 12 und 13 der WCAG 1.0 wiederzufinden.

Prinzip 3
„Inhalte und Kontrollelemente müssen leicht verständlich sein."

Dieser Punkt enthält Richtlinien für leserliche und gut verständliche Textinhalte (Readability), sowie Anregungen zu einer konsistenten und vorhersehbaren Platzierung und Funktionalität von Kontrollelementen.

Hier sind unter anderem inhaltlich die Richtlinien 4 und 14 der WCAG 1.0 wiederzufinden.

Prinzip 4
„Inhalte müssen robust genug sein, um mit aktuellen und zukünftigen Benutzeragenten und assistiven Technologien arbeiten zu können."

Unter dem letzten Hauptpunkt werden also Richtlinien und Checkpunkte zur Sicherung der Kompatibilität gesammelt, um eine möglichst breite Zugänglichkeit sicherzustellen oder zumindest geeignete Alternativen anzubieten.

Hier sind unter anderem inhaltlich die Richtlinien 6, 10 und 11 der WCAG 1.0 wiederzufinden.

Kein direkt nahe liegendes Gegenstück gab es für die Richtlinie 5 („Erstellen Sie Tabellen, die geschmeidig transformieren."). Dies dürfte daran liegen, dass vor allem dieser Punkt sehr HTML-spezifisch formuliert war, was bei der Erstellung der neuen Version vermieden werden sollte. Die Bedeutung der Regel findet sich aber verteilt auf die vier Prinzipien an anderen Stellen wieder [62].

2.3.3 Erfolgskriterien und Konformitätslevel

Wie auch bei den WCAG 1.0, gibt es für jede Regel bestimmte Checkpunkte, die es zu erfüllen gilt. In der Version 2.0 heißen diese Checkpunkte aber Erfolgskriterien und es gibt in Summe 56 davon. Diese sind wieder je nach Dringlichkeit in drei Stufen (Level 1 – 3) unterteilt und die Erfüllung der Erfolgskriterien eines bestimmten Levels resultiert in dem entsprechenden Konformitätslevel A (alle Erfolgskriterien Stufe 1 erfüllt), Double-A (alle Erfolgskriterien Stufe 1 + 2 erfüllt) oder Triple-A (alle Kriterien Stufe 1 + 2 und mindestens 50 % der Erfolgskriterien Stufe 3 erfüllt). Dass für Triple-A nur noch 50 % der Stufe 3 Kriterien erfüllt werden müssen, ist neu im Vergleich zu den WCAG 1.0.

Ebenfalls neu ist die Möglichkeit - in einem so genannten „conformance claim" - nun zum erreichten Konformitätslevel noch zusätzlich erfüllte Checkpunkte eines anderen Levels mit anzugeben, welches aber nicht vollständig erfüllt wird. Damit will man dem Kritikpunkt nachkommen, dass es in der Version 1.0 keinen Anreiz gab, mehr Checkpunkte zu erfüllen als unbedingt nötig. Schließlich gibt es nun noch „baseline"-Definitionen, mit denen man eine Art Min-

destanforderung an Benutzeragenten (und Zusatztechnologien) definieren kann, die von der Seite vorausgesetzt wird und unter deren Voraussetzung sie konform ist [62].

2.3.4 Organisation der Dokumente

Die Version 2 des Regelwerks ist auf drei große Dokumente verteilt: Zuerst gibt es das Dokument der „Web Content Accessibility Guidelines 2.0“ selbst. Es besteht aus einer kurzen Einführung, aus Informationen und Definitionen zur Konformität und aus den eigentlichen Guidelines mit den dazugehörigen Erfolgskriterien. Von dort aus wird direkt zu dem zweiten Dokument „Understanding WCAG 2.0“ [58] gelinkt, wo ausführliche Beschreibungen und Erklärungen zu den einzelnen Regeln sowie Hinweise zur Erfüllung der Kriterien zu finden sind. Beide Dokumente sind technologieneutral formuliert und sollen dadurch eine Anwendbarkeit des Regelwerks auf alle gängigen technischen Spezifikationen sicherstellen. Das dritte große Dokument „Techniques for WCAG 2.0“ [57] beschreibt schließlich konkrete Techniken zur Umsetzung der Erfolgskriterien: Enthalten sind universell anwendbare, generelle Techniken aber nun auch konkrete, auf bestimmte Technologien bezogene Umsetzungsmethoden. Abbildung 2.2 unten veranschaulicht diesen modularen Aufbau.

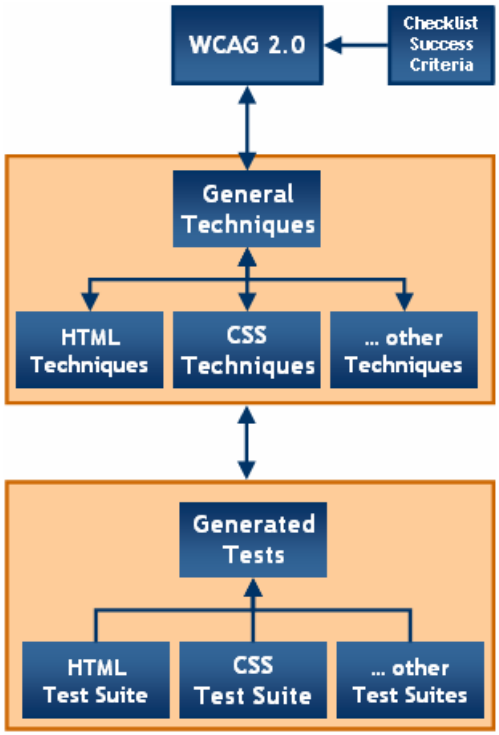


Abbildung 2.2: WCAG 2.0 - Module für Techniken und Tests [49]

2.4 US Rehabilitation Act, Section 508

Der Rehabilitation Act des U.S. Kongresses aus dem Jahr 1973 beschäftigt sich mit der Gleichstellung von physisch und mental Behinderten in der amerikanischen Öffentlichkeit. Es wurde gesetzlich festgehalten, dass diese Personengruppe die selben Rechte hat, wie jeder andere U.S. Bürger und dass deren Selbstständigkeit, Entscheidungsfreiheit und Gleichstellung in Beruf, Bildung und allen anderen gesellschaftlichen Bereichen sichergestellt werden muss. Staatlichen Institutionen wird dabei eine Vorreiterrolle bei der Umsetzung solcher Gleichstellungsmaßnahmen zuteil [44].

1998 wurde das Gesetz überarbeitet, um auf neue Anforderungen in der Informationstechnologie einzugehen. Section 508 enthält seither die Regelungen, um Barrieren im IT-Bereich abzu-

bauen und Technologien zur Gleichstellung zu fördern [19]. Geregelt werden aber nicht nur für das Web (Internet und Intranet) relevante Dinge, sondern auch die Bereiche

- Betriebssysteme und Anwendungssoftware,
- Telekommunikation,
- Video und Multimedia,
- Desktop- und tragbare Computer,
- und eigenständige Systeme (z. B. Kioskanwendungen ohne Schnittstellen für assistive Technologien) [21].

2.4.1 Unterschiede zu den WCAG 1.0

Der Paragraph 1194, Absatz 22, enthält schließlich 16 Regeln (a bis p) zur Accessibility von Internet- und Intranetanwendungen und orientiert sich stark an der Version 1.0 der Web Content Accessibility Guidelines des W3C. Die Regeln a) bis k) decken sich dabei mit Priorität 1 Checkpunkten der WCAG. Damit eine Webseite laut Section 508 das Konformitätslevel A erreicht, müssen allerdings zusätzlich noch folgende Punkte erfüllt werden:

(l) Wenn Interfaceelemente oder Inhalte mit Scriptsprachen dargestellt werden, muss die Information, die durch das Script repräsentiert wird, zusätzlich durch einen entsprechenden Text, der von assistiven Technologien erfasst werden kann, beschrieben werden.

(m) Applets, Plugins oder andere Anwendungen, die auf dem Clientsystem zur Anzeige der Inhalte installiert sein müssen, müssen ebenfalls zu den Section 508 Standards für Betriebssysteme und Anwendungssoftware konform sein.

(n) Online-Formulare müssen assistiven Technologien die volle Funktionalität (Zugang, Auffindbarkeit, Ausfüllbarkeit, …) zur Verfügung stellen.

(o) Navigationszyklen, die sich wiederholen, sollen übersprungen werden können.

(p) Werden Rückmeldungen in einer bestimmten Zeit gefordert, so ist der Benutzer darüber zu informieren und ein Mechanismus zur Anforderung von zusätzlicher Bearbeitungszeit bereitzustellen [20].

Im Gegensatz dazu kommen sämtliche Priorität 2 und 3 sowie die vier folgenden Priorität 1 Checkpunkte der WCAG 1.0 gar nicht in der Section 508 vor. Das Gremium gab zumeist „mangelnde Umsetzbarkeit“ oder „unklare Definitionen“ als Gründe für die Streichungen an:

1.3 Solange Untertitel von Multimediainhalten nicht von den Benutzeragenten vorgelesen werden können, sollen äquivalente Tonaufzeichnungen dafür zur Verfügung gestellt werden.

4.1 Änderungen in der Textsprache müssen klar identifiziert werden.

6.2 Wenn dynamische Inhalte sich verändern, muss auch ein entsprechendes Update der äquivalenten Inhalte dafür sichergestellt sein.

14.1 Es muss eine klare, einfache und angemessene Sprache verwendet werden.

In Summe kann man also sagen, dass die Section 508 deutlich geringere Anforderungen an Webseiten stellt als die WCAG 1.0. Dafür regelt sie aber neben dem Webdesign auch die in Abschnitt 2.4 oben bereits erwähnten, anderen Teilbereiche der IT-Branche. Diese umfangreiche Regulierung, sowie der Umstand, dass viele der größten IT-Firmen ihren Sitz in den USA haben, verleiht der Section 508 besonderes Gewicht, da sich die Anforderungen und Bestellungen der Bundesbehörden eines so großen Staates auch darauf auswirken, wie sich solche Firmen verhalten, was sie produzieren und nach welchen Standards sie arbeiten [15].

2.5 BITV – Barrierefreie Informationstechnik-Verordnung

Die Verordnung zur Schaffung barrierefreier Informationstechnik nach dem Behindertengleichstellungsgesetz regelt in Deutschland seit dem 27. April 2002 die Zugänglichkeitsmaßnahmen für öffentliche Internetauftritte und -angebote und für öffentlich zugängliche graphische Programmoberflächen der Behörden der Bundesverwaltung. Für bereits vorhandene Webangebote endete die Umsetzungsfrist für die in der BITV formulierten Maßnahmen am 31. Dezember 2005. Für neue Webauftritte gelten die Kriterien bereits bei der Erstellung [5].

2.5.1 Unterschiede zu den WCAG 1.0

Wie auch die Bestimmungen der Section 508 in den USA, basiert die BITV auf den Zugänglichkeitsrichtlinien der WCAG 1.0 des World Wide Web Consortium. Geringfügige Änderungen gibt es nur bei der Nummerierung einiger Punkte sowie bei der Einteilung in die Prioritätsstufen: Die BITV fasst die Priorität 1 und 2 Checkpunkte der WCAG zu Priorität I zusammen und BITV Priorität II entspricht der Prioritätsstufe 3 der WCAG 1.0.

Passend zu dieser Einteilung regelt der Paragraph 3 die anzuwendenden Standards so, „dass

- alle Angebote die unter Priorität I aufgeführten Anforderungen und Bedingungen erfüllen und
- zentrale Navigations- und Einstiegsangebote zusätzlich die unter Priorität II aufgeführten Anforderungen und Bedingungen berücksichtigen.“ [4]

2.5.2 Status der Umsetzung

BIK (barrierefrei informieren und kommunizieren) ist ein dreijähriges Gemeinschaftsprojekt aus deutschen Blinden- und Sehbehindertenverbänden sowie der DIAS GmbH (www.dias.de) mit dem Ziel, Webangebote besser zugänglich für behinderte Menschen zu machen und dies auch zu überprüfen. In der ersten Jahreshälfte 2006 wurde deshalb eine umfassende Aktion gestartet, um zu erheben, wie weit die Umsetzung der BITV nach Ablauf der Übergangsfrist bereits fortgeschritten ist [32]. 24 an Ministerien, Behörden und Versicherungen durchgeführte „Tests der Woche“ brachten dabei folgendes Ergebnis:

- 5 Mal „gut oder sehr gut zugänglich“ (90 bis 100 von 100 möglichen Punkten)
  - ✓ Die Gesundheitskarte, http://www.die-gesundheitskarte.de/
  - ✓ BRAMER Ersatzkasse, http://www.barmer.de/
  - ✓ bio-siegel.de, http://www.bio-siegel.de/
  - ✓ Techniker Krankenkasse, http://www.tk-online.de/
  - ✓ Bundeszentrale für gesundheitliche Aufklärung, http://www.bzga.de/
- 11 Mal „eingeschränkt zugänglich“ (80 bis 89 Punkte)
- 8 Mal „schlecht zugänglich“ (weniger als 80 Punkte)

Die eingeschränkt zugänglichen Seiten wiesen dabei vor allem Probleme bezüglich der Zugänglichkeit von PDFs, Hindernisse für blinde und sehschwache Personen (schlechte Strukturierung der Inhalte, ungeeignete Kontraste und Schriftgrößen) oder allgemein eine schlechte Tastaturbedienbarkeit auf. Als nicht akzeptabel wurde vor allem das „schlecht zugängliche“ Ergebnis der beiden großen Betriebskrankenkassen (Deutsche BKK, Bertelsmann BKK) dargestellt, da gerade deren Webservices für Millionen behinderte, kranke und ältere Menschen wichtig sind [30].

Neben den 24 ausführlichen Tests wurden im August dieses Jahres an 112 weiteren Webangeboten standardisierte Vortests durchgeführt, um ein besseres (größeres) Gesamtbild zu erhalten. Die folgende Tabelle 2.1 zeigt die Gesamtergebnisse der zwölf überprüften Kriterien:

Kriterium	erfüllt	teilweise erfüllt	nicht erfüllt
brauchbare Alternativtexte	61 %	28 %	11 %
keine Schriftgrafiken	64 %	22 %	14 %
Code valide	30 %	13 %	57 %
keine Layouttabellen	42 %	11 %	47 %
Schriften vergrößerbar	67,5 %	10,25 %	22,25 %
Inhalte strukturiert	24 %	55,5 %	20,5 %
linearisierbar	90,5 %	4,25 %	5,25 %
ohne Skripte nutzbar	86,25 %	6 %	7,75 %
kein Blinken/Bewegung	84,5 %	9,5 %	6 %
tastaturbedienbar	84,5 %	8,5 %	7 %
keine veralteten Elemente	25,5 %	40 %	34,5 %
keine separate Textversion	89 %	0 %	11 %

Tabelle 2.1: Ergebnisse der BIK Vortests nach Kriterien [31]

Die größten Mängel fanden sich also in der Verwendung von fehlerhaftem oder veraltetem Code, in der ungenügenden Struktur des Dokuments (Trennung von Layout und Struktur) und im immer noch häufigen Einsatz von Tabellen zu Layoutzwecken.

Überträgt man die Ergebnisse der vereinfachten Vortests tendenziell auf die Bewertungsskala der ausführlichen „Tests der Woche“, dann können 25 der überprüften Webangebote (21,5 %) „gut oder sehr gut zugänglich“ erreichen, 50 Seiten (43 %) tendieren zu „eingeschränkt zugänglich“ und die restlichen 41 Webauftritte (35,5 %) bleiben „schlecht zugänglich“. Dies deckt sich in etwa mit der Verteilung der Ergebnisse der vollständigen BITV-Tests und lässt also durchaus einen Schluss auf die Gesamtsituation in Deutschland zu [31].

2.6 Diskussion und Schlussfolgerung

Keine einheitliche Regelung

Der Anfang aller Probleme im Accessibility-Bereich liegt darin, dass es keine weltweit einheitliche, verbindliche Richtlinie gibt. Die Empfehlungen des W3C stellen zwar eine Art internationalen Standard dar, sind aber eben keine Norm und nicht gesetzlich verpflichtend, weshalb man überall auf der Welt unterschiedliche nationale Regulierungen vorfindet. Diese mögen zwar alle ähnlich sein und in vielen Bereichen auf den W3C Recommendations basieren, es finden sich aber sehr häufig Zusätze, Ungenauigkeiten oder Abstriche.

Dadurch stehen Webautoren vor dem Dilemma, dass sie mit mehreren gängigen Regelwerken vertraut sein müssen, wenn sie einerseits Inhalte erstellen wollen, die konform zu der „globalen“ Empfehlung des W3C sein sollen und andererseits abweichende Bestimmungen des nationalen Gesetzgebers erfüllen müssen. Auch für Hersteller von assistiver Software, Autorenwerkzeugen oder Prüfwerkzeugen ergeben sich diese Probleme: Auf die Einhaltung welcher Normen kann man sich verlassen? Welche müssen geprüft werden? Was ist als Fehler zu werten und was nicht? Solche Werkzeuge müssen also entweder die verschiedensten Regelwerke unterstützen oder man braucht je nach nationaler Gesetzgebung wieder eigene Werkzeuge um konforme Inhalte zu erstellen oder um auf Konformität zu prüfen. Ganz zu schweigen davon, dass eine Seite aus den USA, welche die Section 508 Standards erfüllt, möglicherweise aber nicht uneingeschränkt zugänglich für einen Besucher aus Deutschland ist, dessen Systemkonfiguration sich

auf die für ihn national gültigen Bestimmungen der BITV verlässt. Das World Wide Web ist zwar ein „weltweites“ Netz, aber offensichtlich doch noch weit entfernt von „write once, read anywhere“.

Fehlendes Bewusstsein für den Nutzen von Accessibility

Der Umstand, dass selbst die nationalen Regulierungen nur für Verwaltungsbehörden verpflichtend sind, ist ein weiterer Kritikpunkt. Private Webseiten oder Firmen können bestenfalls auf freiwilliger Basis zu Accessibility-Maßnahmen angehalten werden. Offensichtlich gibt es aber noch kein ausreichend großes Bewusstsein oder Verständnis wie wichtig Barrierefreiheit ist. Deshalb sieht es mit gut zugänglichen Webangeboten, die nicht aus dem regulierten, staatlichen Bereich kommen, düster aus: Im Juli 2004 wurde eine Studie des eAccessibility Lab des Research Centre for Networks and Communications Engineering der Universität Dublin veröffentlicht, die das Ergebnis eines groß angelegten, automatisierten Accessibility-Tests von Webseiten aus ganz Europa beschreibt. Dabei wurden in Summe ca. 5000 irische, britische, deutsche und französische Webseiten aus den Bereichen

- Kunst und Unterhaltung
- Wirtschaft
- Bildung
- Verwaltung
- Gesundheit
- Medien
- Sport und Freizeit
- Wissenschaft
- Gesellschaft und Kultur
- Beförderungswesen

mit dem Accessibility-Evaluator Bobby getestet. Zur Beurteilung wurden nur die 25 Tests (bzw. Checkpunkte) herangezogen, die von dem Prüfwerkzeug selbstständig und ohne menschliche Entscheidungshilfe verlässlich bewertet werden konnten. Im Schnitt scheiterten dabei 95 % der getesteten Webseiten an den Anforderungen der minimalen Konformitätsklasse WCAG-A. Keine einzige war konform zu WCAG-Triple-A. Zusätzlich wurde auch noch die Validität des Quellcodes überprüft und aus der gesamten Stichprobe hatten nur besorgniserregende 10(!) Seiten vollständig korrektes HTML-Markup. Das waren weniger als 0,15 % aller Seiten [22].

Gratwanderung zwischen Universalität und Umsetzbarkeit

Ein zentraler Kritikpunkt an den WCAG 1.0 war, dass diese zu sehr auf HTML ausgelegt waren. Beim Verifizieren von anderern Sprachen und Technologien hatte man damit Probleme, festzustellen was nun konform ist und was nicht, da es keine passenden Regeln dafür gab. Deswegen wurde in den Anforderungen für die Version 2 festgehalten, dass auf eine generische, technologieabhängige Formulierung geachtet werden muss und die Konformitätskriterien noch klarer definiert werden müssen. Diese neutrale Formulierung wird mit den WCAG 2.0 sicherlich erreicht. Die Umstrukturierung der Regeln in vier zentrale Bereiche kann auch positiv betrachtet werden und erlaubt dem Leser, sich schneller einen groben Gesamteindruck des Regelwerks zu verschaffen.

Allerdings gibt es auch eine Menge Kritikpunkte und heftige Diskussionen in Expertenkreisen zu den WCAG 2.0. So scheint man auf Kosten dieser Universalität und einer ausgefeilten (böse Stimmen sagen „weltfremden“) Formulierung und Terminologie, Bedenken zur konkreten Umsetzbarkeit über Bord geworfen zu haben. Das Regelwerk ist in seinem letzten Entwurf nämlich auf über 450 gedruckte Seiten angewachsen und verliert sich in einer Fülle von Begriffsdefinitionen und widersprüchlichen oder unklaren, abstrakten Formulierungen. Wenn man das Thema

„Barrierefreiheit“ wirklich einer breiten Masse näher bringen will, wird man sich neben aller Generik auch damit beschäftigen müssen, wie man dem kleinen Webdesigner die konkrete Umsetzung und Prüfung der Maßnahmen schnell und einfach verständlich machen kann. Ein Artikel des Accessibility-Beraters Joe Clark titelt wörtlich „Zur Hölle mit den WCAG 2“ [6] und befasst sich sehr kritisch mit vielen dieser Unstimmigkeiten und schwammigen Formulierungen in dem Regelwerk.

Zusammenfassend bleibt zu sagen, dass für sinnvolle Ergebnisse im Web Accessibility-Bereich eine weltweit möglichst einheitliche Regelung angestrebt werden muss, die zudem über eine alleinige Regulierung von staatlichen Institutionen hinausgeht und größere Bereiche des Webs zu erreichen versucht. Dies wird im großen Rahmen wiederum nur funktionieren, wenn Zugänglichkeitsrichtlinien verständlich und konkret umsetzbar gestaltet werden.

3 Accessibility-Evaluatoren

Das Kapitel 3 beschäftigt sich mit Programmen zur automatischen Überprüfung von Zugänglichkeitsrichtlinien, so genannten Accessibility-Evaluatoren. Es werden einige bekannte Freeware Tools vorgestellt und deren Eigenheiten herausgearbeitet und Bereiche identifiziert, die verbesserungswürdig sind. Danach folgt noch eine kurze Einführung in die Evaluation and Report Language, die künftig für eine normierte Ergebnisausgabe von Accessibility-Evaluatoren sorgen soll. Abschließend werden die gefundenen Ergebnisse zusammengefasst und diskutiert.

Bei der Betrachtung der Evaluatoren soll weder deren eigene Usability, noch die Art der Präsentation der Ergebnisse im Vordergrund stehen. Primär geht es darum, Bereiche zu identifizieren, wo die Analyseverfahren verbessert werden könnten, um mehr Tests zu automatisieren und so die Anzahl der manuellen Prüfschritte zu reduzieren. Alle im Folgenden betrachteten Tools sind frei zugänglich und nicht kostenpflichtig. Dies hat den einfachen Grund, dass Web-Accessibility ein globales Thema ist und geeignete Prüfsoftware nicht nur in Form von teuren, kommerziellen Produkten für große Firmen oder Institutionen zur Verfügung stehen sollte, sondern auch den vielen kleinen und privaten Webautoren eine verlässliche und kostenlose Möglichkeit geboten werden muss, ihre Inhalte zu prüfen. Accessibility kann nicht nur gefordert werden, sie muss gefördert werden! Und die Zugänglichkeit zu guter Prüfsoftware von teuren Lizenzen abhängig zu machen, stellt bereits das erste Hindernis von umfassender Barrierefreiheit dar.

3.1 WebXACT

WebXACT ist der Nachfolger des populären, mittlerweile kommerziellen Accessibility-Tools Bobby und wird von der Firma Watchfire angeboten, die unter anderem auch Software zur Prüfung von Sicherheits- und Datenschutzthemen entwickelt. Es ist ein serverseitiges Tool mit Webinterface und unter folgender Adresse zu erreichen: http://webxact.watchfire.com/

Überprüft werden können jeweils einzelne Seiten auf Übereinstimmung mit den Zugänglichkeitskriterien der Section 508 und den Web Content Accessibility Guidelines 1.0. Optional kann man die gewählte Seite auch auf defekte Links testen und sich die Codefragmente anzeigen lassen, in denen das Tool Probleme festgestellt hat (vergleiche Abbildung 3.1 unten).

Das Ergebnis eines Scans wird dann unterteilt in vier Teilbereiche dargestellt: Unter „General“ zeigt das Programm eine kurze Übersicht allgemeiner Daten wie die Dateigröße der Seite, die verwendeten Metadaten, die Anzahl der Bilder oder den Zeitpunkt der letzten Änderung an. Der Punkt „Quality“ beschäftigt sich mit der Anzahl defekter Links, der Anzahl fehlender Breiten- und Höhenangaben und ähnlich einfachen Test, die am Quellcode durchgeführt werden. Unter „Privacy“ werden dann noch ein paar einfache Überprüfungen zum Thema Datenschutz zusammengefasst. Wie zum Beispiel der Verschlüsselungslevel der Seite, die Anzahl der Formulare, die GET verwenden, oder Anzahl und Art der Cookies, welche die Seite setzen wollte. Im Accessibility-Abschnitt findet man schließlich eine genaue Aufschlüsselung der Zugänglichkeitsprobleme, die WebXACT identifiziert hat.

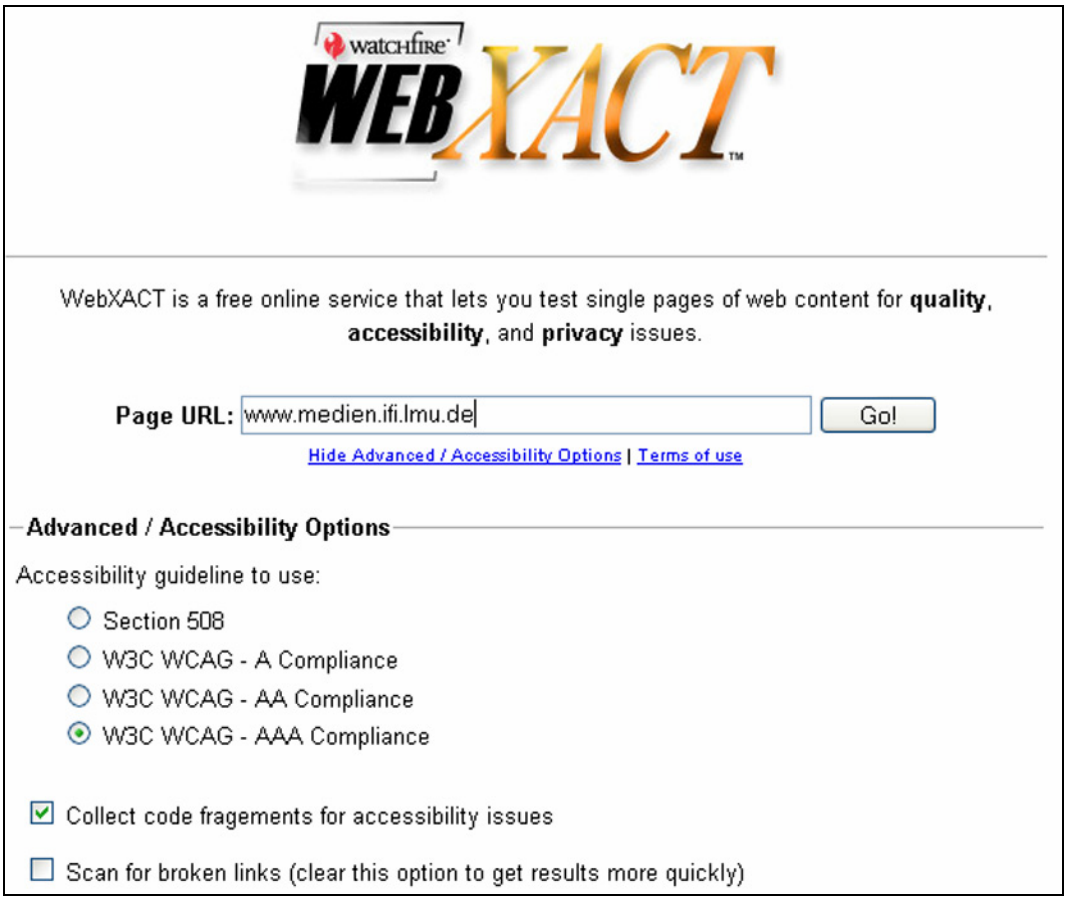


Abbildung 3.1: Startseite von WebXACT

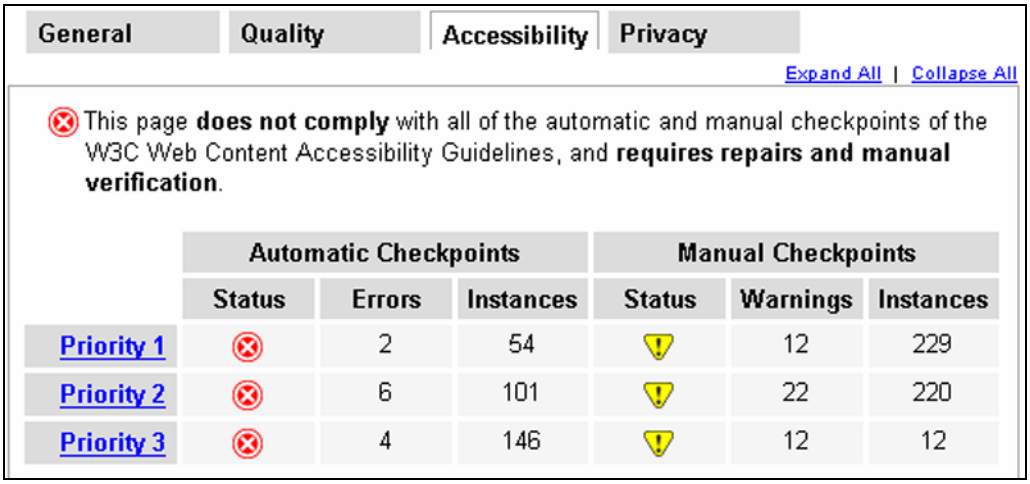


Abbildung 3.2: WebXACT - Übersicht der Accessibility-Probleme

Abbildung 3.2 oben zeigt die Ausgabe des Tools nach der Überprüfung einer Seite. Die gefundenen Probleme werden nach Priorität und nach der Art des Checkpunkts unterteilt. Das Programm unterscheidet zwischen automatischen Checkpunkten, die es mit Sicherheit eigenständig überprüfen kann (Error) und manuellen Checkpunkten, wo das Programm eine Verletzung der Richtlinie für möglich hält, der Benutzer aber händisch nachprüfen muss (Warning). Man erkennt sofort, dass die Anzahl der Probleme, die manuelle Nachprüfungen erfordern, weit größer ist, als die Anzahl der Fehler, die eindeutig identifiziert werden konnten. Dies ist ein generelles Problem von Accessibility-Evaluatoren, da für sehr viele Checkpunkte die Semantik und der

Kontext der Inhalte ausschlaggebend sind, was wiederum von Maschinen ohne menschliche Hilfe nicht oder nur sehr eingeschränkt erfasst werden kann.

Unterhalb der Übersicht befindet sich eine detaillierte, tabellarische Aufstellung der Fehler und Warnungen, wiederum sortiert nach Priorität (vergleiche Abbildung 3.3 unten). Zu jedem Problem werden die entsprechende Richtlinie (inkl. Nummer), die Anzahl der Vorkommen auf der geprüften Seite sowie die dazugehörigen Codezeilen angezeigt. Optional kann man sich diese Codezeilen auch noch anzeigen lassen. Klickt man auf den Namen der Richtlinie, dann öffnet sich ein Fenster mit einer umfassenden Erklärung des Checkpunkts und Hinweisen wie man diesen umsetzen kann.

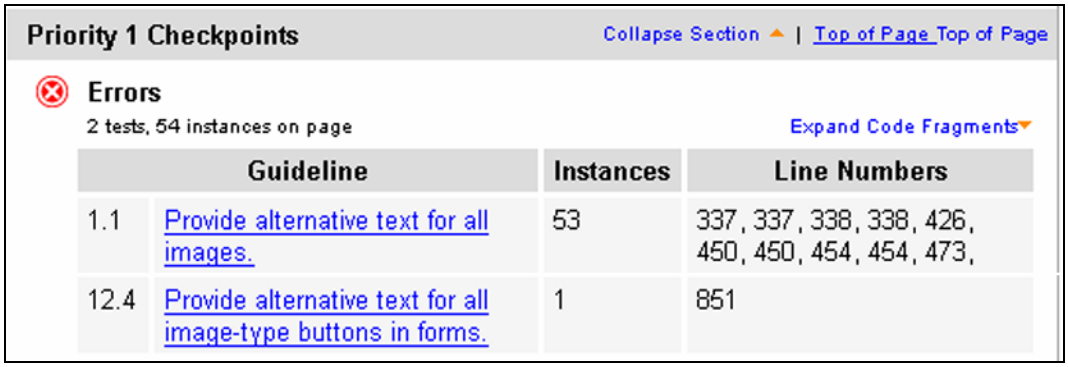


Abbildung 3.3: WebXACT - Detailanzeige der Fehler

3.2 WAVE 3.0

Die gemeinnützige Initiative WebAIM (Web Accessibility in Mind) ist eine Zusammenarbeit des Center for Persons with Disabilities und der Utah State University mit dem Ziel, das Potential des Webs für behinderte Personen zu vergrößern. Zu diesem Zweck wurde neben vielen anderen Aktivitäten auch ein Werkzeug zur Accessibility-Überprüfung entwickelt: WAVE 3.0. Wiederum handelt es sich um ein kostenloses Online-Tool, welches unter http://wave.webaim.org/ zu finden ist.

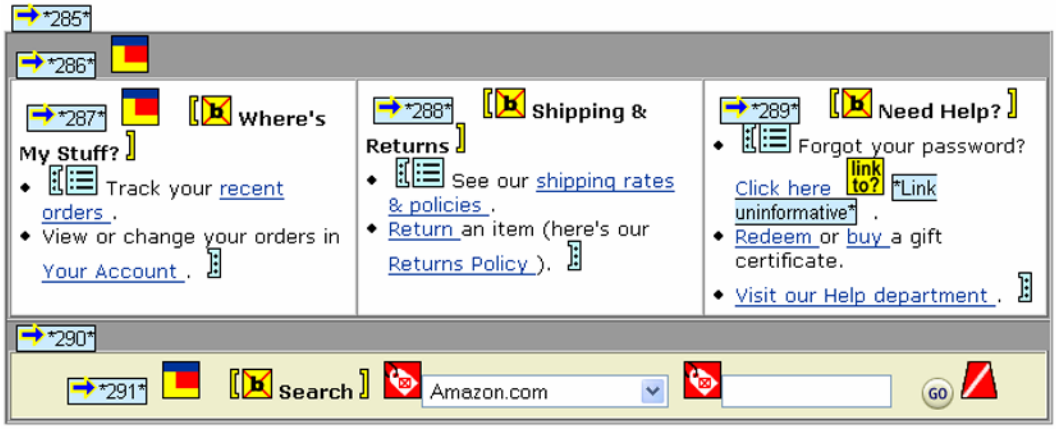


Abbildung 3.4: WAVE - Eine mit Symbolen annotierte Webseite

WAVE bietet neben der Möglichkeit eine Seite über die Übermittlung ihrer URI zu prüfen, auch noch den direkten Upload von Dateien sowie eine Browsertoolbar und den Start über ein Lesezeichen im Browser an. Auch hier werden die Standards des W3C und der Section 508 unterstützt. Zusätzlich kann man noch konfigurieren, welche Elemente der Originalseite in der Auswertung angezeigt werden sollen und welche Features von WAVE (Fehler, Warnungen, Rah-

men um Bereiche, Legende der Symbole, …) diese beinhalten soll. Das Ergebnis der Evaluierung wird, im Gegensatz zu der rein textbasierten Ausgabe von WebXACT, nämlich in Form von Symbolen direkt in die Ansicht/den Code der geprüften Seite eingefügt (vergleiche Abbildung 3.4 oben).

Eine Legende der verwendeten Symbole inklusive näheren Erklärungen und empfohlenen Maßnahmen kann man in einem eigenen Fenster öffnen. Die Farbgebung der Symbole richtet sich dabei nach folgender Einteilung:

- Rote Symbole stehen für Fehler.
- Gelbe Symbole stehen für Warnungen, die näher überprüft werden sollten.
- Grüne Symbole repräsentieren Accessibility-Features (auf korrekte Verwendung überprüfen).
- Hellblau werden Bereiche mit strukturellen, semantischen oder Navigationsinhalten dargestellt (auf korrekte Verwendung überprüfen).
- Alle anderen Farben haben keine spezifische Bedeutung.

3.2.1 WAVE 3.5 (Entwicklungsversion)

Die noch in Entwicklung befindliche Nachfolgeversion WAVE 3.5 kann bereits auf den WebAIM-Seiten getestet werden und bietet einige interessante Neuerungen bei der Ausgabe der Ergebnisse (vergleiche Abbildung 3.5 unten):

- Im Dokumentkopf befindet sich nun eine Zusammenfassung mit der Anzahl der gefundenen Fehler, Warnungen, … .
- Fährt man mit der Maus über eines der Symbole in der Seite, so erscheint ein Tooltip-Text, der die Bedeutung des Symbols erklärt.
- In einem separaten Bereich wird eine reine Textansicht der analysierten Seite angezeigt (ohne CSS, ohne Skripte, ohne Grafiken), in der der Benutzer überprüfen kann, ob eine sinnvolle Lesereihenfolge beibehalten wird, ob passende Alternativtexte für die Bilder verwendet wurden und ob keine Informationen verloren gehen, die sich allein auf Farben, Skripte oder sonstige externe Hilfsmittel gestützt haben.
- Ein weiterer Bereich zeigt eine verschachtelte Gliederungsansicht aller Überschriftentags (H1-H6) - zur Kontrolle, ob diese richtig verwendet wurden.
- Zusätzlich kann man sich die Ergebnisse des Tests auch im EARL-Format (siehe Kapitel 3.5) ausgeben lassen.

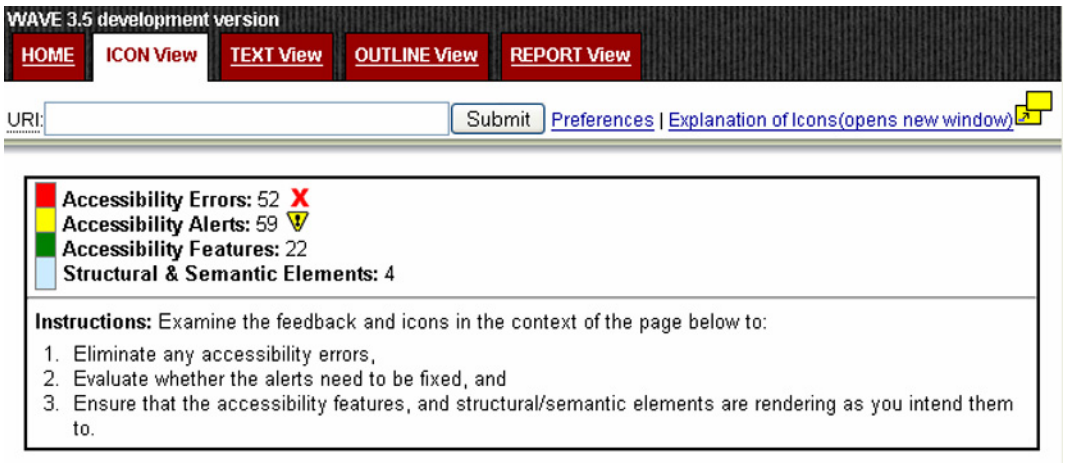


Abbildung 3.5: WAVE - Verschiedene Ansichten des Testergebnisses

3.3 A-Prompt

Ein weiteres Tool zur Verifizierung von Accessibility-Richtlinien ist A-Prompt (Accessibility Prompt), das von dem Adaptive Technology Ressource Centre (ATRC) der Universität Toronto entwickelt wurde. Im Gegensatz zu den bereits vorgestellten Werkzeugen, muss man A-Prompt installieren und es ist nur für Windows-Plattformen verfügbar, dafür kann es auch als DLL direkt in HTML-Autorenwerkzeuge eingebunden werden und ermöglicht halbautomatische Reparaturen im Quellcode. Unterstützt werden wiederum die WCAG und die Section 508 Standards (Downloadmöglichkeit: http://aprompt.snow.utoronto.ca/).

Während der Entwicklung von A-Prompt wurde auf drei grundsätzliche Punkte Wert gelegt:

- So viele Tests wie möglich sollten automatisiert werden, damit der Benutzer sich auf den Rest konzentrieren kann.
- Es wurde eine größtmögliche Integration mit Autorenwerkzeugen angestrebt, um Accessibility-Probleme bereits während des ersten Seitendesigns zu vermeiden.
- Benutzer des Tools sollen über Hilfen, Beschreibungen und Ratschläge quasi unterbewusst für ein zugängliches Design geschult werden [34].

Abbildung 3.6 unten zeigt, wie A-Prompt den Benutzer durch eine automatische Reparatur führt.

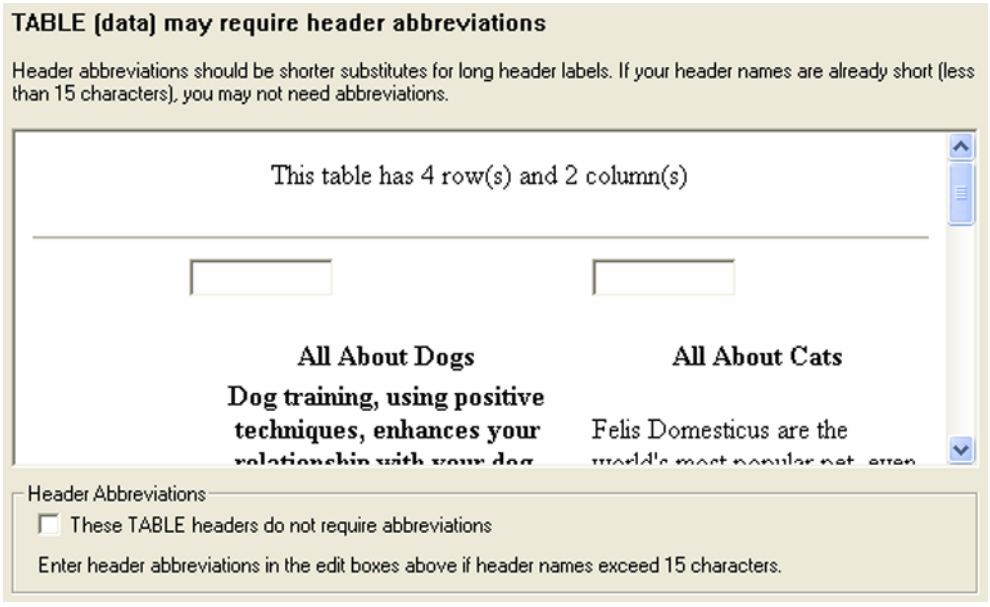


Abbildung 3.6: A-Prompt - Automatische Reparatur

3.4 ATRC Web Accessibility Checker

Kurz vor Ende dieser Arbeit wurde beim Herunterladen einiger Informationen von der A-Prompt Webseite eine Weiterleitung auf eine Nachfolgeversion beziehungsweise eine eigenständige Weiterentwicklung der A-Prompt-Software gefunden. Das ATRC gab bekannt, dass die Betreuung von A-Prompt in absehbarer Zeit komplett zugunsten der neuen Software eingestellt wird und der neue Web Accessibility Checker (http://checker.atrc.utoronto.ca/) von ATRC die Erkenntnisse aus dem A-Prompt-Projekt weiterverwenden wird und zusätzlich eine Menge an Verbesserungen und Neuerungen bietet:

- Zusätzlich zu den gängigen Regelwerken (Section 508 und WCAG 1.0) werden auch bereits die WCAG 2.0 sowie weitere nationale Richtlinienkataloge unterstützt: die deutsche BITV und der italienische Stanca Act.
- Es werden umfangreichere Accessibility-Tests als in A-Prompt durchgeführt.
- Ergebnisberichte können bereits im EARL-Format ausgegeben werden.
- Der neue Checker ist nun webbasiert und damit plattformunabhängig.

Und tatsächlich ist dieses Programm im Vergleich zu vorangegangenen Tools ein Musterbeispiel an Modularität und sauber organisierten Tests, die sich exakt an die Vorschläge und Abläufe des W3C halten. Das ist wohl dadurch zu begründen, dass einige Mitarbeiter des ATRC auch in Arbeitsgruppen des W3C (z. B. eben in der WCAG 2.0 Arbeitsgruppe) tätig sind. So sind beispielsweise sämtliche unterstützte Richtlinienkataloge in Form von XML-Dateien gespeichert, wo zu jeder Regel eine Liste von einzelnen Prüfschritten (gemäß WCAG) angegeben ist, die von dem Prüfprogramm dann abgearbeitet werden. Die Datei checks.xml beinhaltet die komplette Liste dieser Accessibility-Checks inklusive detaillierter Fehlerbeschreibungen, Reparaturanweisungen, Listen von Folgechecks und Anweisungen für die Prüfmaschine, wie der Check auszuführen ist (<machine>…</machine>). Auflistung A.1 im Anhang dieser Arbeit zeigt einen Ausschnitt aus dieser XML-Datei.

Wenn man sich auf der Checker-Webseite registriert, erhält man zusätzlich die Möglichkeit, während der Auswertung seiner Testergebnisse Entscheidungen (decisions) zu treffen. Abbildung 3.7 unten zeigt die Möglichkeit eine solche Entscheidung auszuführen, die sich dann entsprechend auf nachfolgende Checks auswirkt und dazu beiträgt, dass im fertigen Accessibility-Bericht (z. B. in der EARL-Ausgabeversion) keine überflüssigen Warnhinweise mehr vorkommen. Im unteren Bereich der Grafik sieht man außerdem noch Anforderungen, Testprozess und erwartetes Ergebnis des Tests, so wie sie in checks.xml für diesen Testfall definiert wurden und hier zur Veranschaulichung für den Benutzer ausgegeben werden.

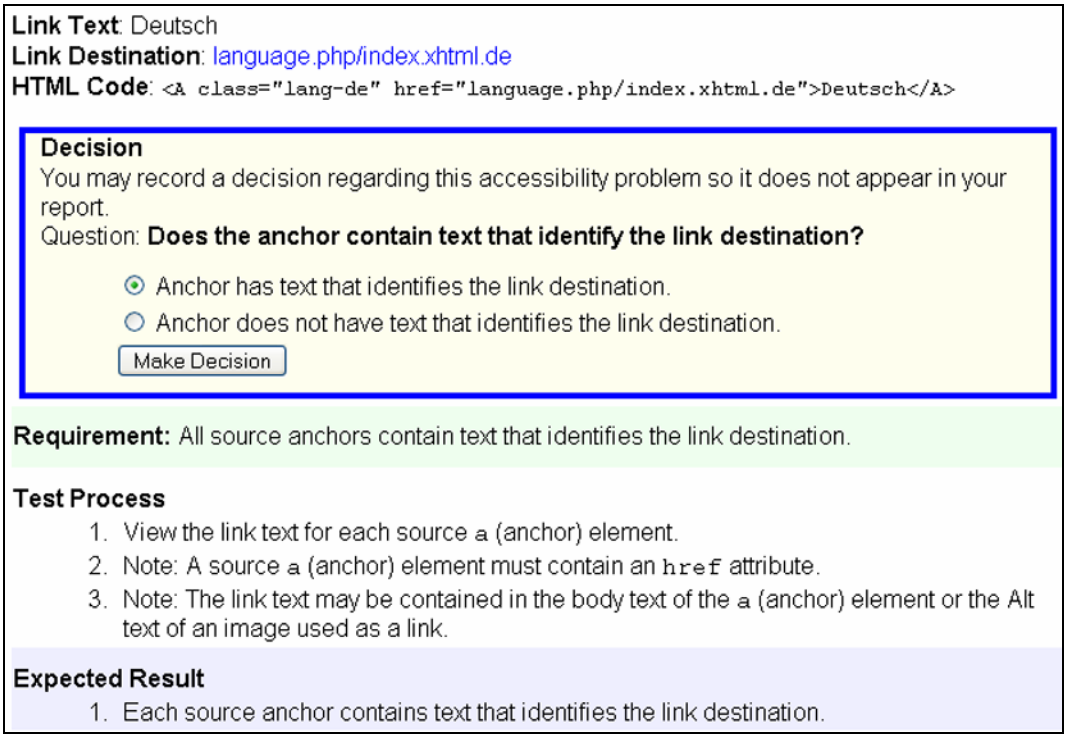


Abbildung 3.7: Eine Entscheidungsmöglichkeit beim ATRC Checker

Der Checker ist zudem ein Open Source Projekt und darf völlig frei verwendet, kopiert, modifiziert oder weitergegeben werden. Wie einleitend bereits erwähnt wurde, wurde diese Software allerdings erst kurz vor Ende dieser Arbeit entdeckt, ansonsten hätte man versuchen können, die hier vorgestellten Erkenntnisse und Verfahren direkt in den ATRC Checker zu integrieren. So hätte man die prototypischen Umsetzungen der nachfolgenden Kapitel bereits auf vollständigen Regelwerken demonstrieren können. Zukünftige Projekte auf diesem Gebiet sollten auf jeden Fall die Entwicklung des ATRC Accessibility Checker im Auge behalten.

3.5 Evaluation and Report Language (EARL)

EARL ist ein Dialekt des Ressource Description Framework (RDF) [55][56] und soll dabei helfen, Daten über die Auswertung von Ressourcen speichern, weitergeben und weiterverarbeiten zu können. Das W3C versucht damit eine einheitliche Repräsentation von Accessibility-Testergebnissen zu erreichen, um diese besser vergleichen zu können. So sollen zu jedem Test genaue Informationen über

- den Kontext (Wann und womit wurde bzw. wer hat getestet?)
- das Testsubjekt (Was wurde getestet?)
- das Ergebnis/die Gültigkeit (Wie lautet das Ergebnis und wie sicher ist es?)
- die Testkriterien (Auf welche Kriterien wurde getestet?)

festgelegt und gespeichert werden [52]. Es handelt sich also um kein Accessibility-Tool mehr, trotzdem wurde dieser Unterpunkt hier angeführt, da neuere Tools bereits alle Bezug auf EARL nehmen und kurz erklärt werden sollte, worum es sich handelt. Auflistung 3.1 unten zeigt einen Ausschnitt eines Dokuments im EARL-Format, der einen gescheiterten Testfall beschreibt.

```
1  <earl:result rdf:ID=”result”>
2    <earl:validity rdf:resource=”http://www.w3.org/WAI/ER/EARL/nmg-
   strawman#fail”/>
3    <dc:title xml:lang=”en”>Invalid Markup (code #353)</dc:title>
4    <dc:description rdf:parseType=”Literal” xml:lang=”en”>
5      <div xmlns=”http://www.w3.org/1999/xhtml”>
6        <p>The <code>table</code> element is not allowed to appear
7        inside a <code>p</code> element</p>
8      </div>
9    </dc:description>
10   <earl:instance rdf:resource=”#instance”/>
11 </earl:result>
```

Auflistung 3.1: EARL-Beispielausgabe für einen gescheiterten Test

3.6 Diskussion und Schlussfolgerung

Der Hauptkritikpunkt an automatischen Accessibility-Evaluatoren ist der Umstand, dass nur sehr wenige Tests wirklich automatisch durchgeführt werden können. Laut einem Artikel von Karl Dawson auf Accessites.org [10] können von allen 65 Checkpunkten der WCAG 1.0 sogar nur die folgenden drei gänzlich automatisch überprüft werden:

- „Erstelle Dokumente, die gegen veröffentlichte, formale Grammatiken validieren.“ (Checkpunkt 3.2, Priorität 2)
- „Vermeide die Verwendung von veralteten/überholten W3C-Technologien.“ (Checkpunkt 11.2, Priorität 2)

- „Identifiziere die Primärsprache einer Seite.“ (Checkpunkt 4.3, Priorität 3)

Kein Wunder also, dass viele Kritiker solcher Prüfprogramme zu Recht deren Nutzen anzweifeln. Neben den wenigen, klar entscheidbaren Testfällen unterstützt Accessibility-Software den Benutzer aber zumindest bei einer organisierten, vollständigen Abarbeitung der Checkpunkte und bietet oft auch weitere Hilfen und Reparaturvorschläge an.

Um die Testfälle zu identifizieren die automatische Prüfsoftware vor Probleme stellt, wurden die Programme bei der Durchführung dieser Arbeit, zusätzlich zu den Erkenntnissen von Dawson, mit präparierten HTML-Dateien getestet. Eine Sammlung solcher Testdateien findet man beispielsweise auf den Seiten des HTML Tidy Accessibility Checkers [43]. Auch im Rahmen der WCAG 2.0 finden sich Sammlungen solcher Tests.

Folgende Warnungen, die zu den teilweise beziehungsweise gar nicht automatisierten Fällen gehören, können unserer Meinung nach verbessert werden:

- Warnungen, die aufgrund von fehlenden Kontextinformationen entstehen (z. B. der Typ von Bildern oder Tabellen)
- Warnungen, die die Lesbarkeit der Webseite betreffen
- Warnungen, die die Struktur der Seite (des Markups) betreffen

Eine Verbesserung drückt dabei nicht unbedingt eine vollständige Automatisierung aus, sondern kann auch über Hilfestellungen erreicht werden, die es dem Benutzer ermöglichen die Warnmeldungen schneller und sicherer zu bearbeiten. Im Kapitel 5 dieser Arbeit wird erörtert, inwiefern sich Verbesserungen in diesen Bereichen auch konkret umsetzen lassen.

4 Readability

Das Kapitel Readability beschäftigt sich einleitend mit einigen Grundsätzen zur Lesbarkeit und stellt dann bekannte Readability-Formeln mit ihren Eigenheiten vor. Danach wird die Möglichkeit betrachtet, den Wortschatz von Texten in die Lesbarkeitsüberprüfung mit einzubeziehen. Auch der Einfluss von Farben und daraus resultierenden Kontrasten auf die Lesbarkeit wird in diesem Abschnitt besprochen. Am Ende des Kapitels werden Programme zur automatischen Readability-Prüfung vorgestellt und alle Unterpunkte noch einmal zusammenfassend diskutiert.

Unter Readability (Lesbarkeit) versteht man eine Größe, die beschreibt, wie zugänglich ein Schriftstück für eine (möglichst breite) Leserschaft ist. Zugänglichkeit muss hier im Sinne der Möglichkeit einen Text wahrzunehmen und ihn zu verstehen, gesehen werden. Die Lesbarkeit eines Textes hängt abgesehen von individuellen Voraussetzungen des Lesers (Sprachfähigkeiten, Fachkenntnisse, Interessen, Konzentration, …) grundsätzlich von drei Faktoren ab:

1. Erkennbarkeit (Typographie)

   Zur Erkennbarkeit tragen typographische Merkmale wie die Schriftart, -größe und -farbe sowie Zeilenabstände, Wortzwischenräume aber auch das Ausgabegerät (Papier vs. Monitor, Größe und Qualität des Displays usw.) bei. Handelt es sich um digitale Texte, so kommt all diesen Kriterien sogar eine noch stärkere Bedeutung zu, da gängige Bildschirme noch immer relativ deutliche Nachteile im Vergleich zu Papierausdrucken haben (Auflösung, Kontrast, Bildwiederholungsfrequenzen).

   Verständlicherweise bevorzugen die meisten Menschen 12 pt Schriftgrößen gegenüber auch sehr häufig vorkommenden Texten mit der kleineren Schriftgröße 10 pt. Schriftarten mit Serifen (Times New Roman) werden als angenehmer empfunden, schneller erfasst und praktisch im gesamten Printmediensektor verwendet. Kinder und ältere Personen hingegen bevorzugen serifenlose Schriften (Verdana) und auf Monitoren geht der allgemeine Trend sowieso zu einfachen, möglicherweise weniger „schönen“, dafür klar erkennbaren Schriftarten ohne Serifen.

   Auf Farben und Kontraste wird später in diesem Kapitel noch einmal detaillierter eingegangen. Messen kann man die Erkennbarkeit ganz einfach über die Lesegeschwindigkeit und die Fehlerhäufigkeit beim Lesen eines Textes durch Probanden [35].

2. Verständlichkeit (Satzbau und Sprachstil)

   Für die Verständlichkeit eines Textes spielen der Anteil an unbekannten Wörtern (seltene Wörter oder Fachbegriffe) sowie die Struktur des Textes eine große Rolle. Texte mit komplizierten, verschachtelten Sätzen und sehr langen Wortkonstruktionen sind schwerer verständlich als Schriftstücke, die vorwiegend kurze, überschaubare Sätze im Aktiv und einfache, geläufige Wörter verwenden.

3. Nachvollziehbarkeit (Komplexität)

   Darunter versteht man, ob und wie schnell sich dem Leser die Bedeutung des Textes erschießt [47].

In den 90er Jahren ergaben mehrere Studien in den USA, dass der tödliche Ausgang vieler Verkehrsunfälle mit Kindern mit der unsachgemäßen Handhabung von Kindersitzen zusammenhängt. Spezialisten der Gesundheitsbehörde führten daraufhin mit Hilfe der SMOG-Lesbarkeitsformel (siehe Kapitel 4.1.3) eine Untersuchung der Lesbarkeit von Bedienungsanleitungen solcher Sicherheitssitze durch und kamen zu einem erschreckenden Ergebnis:

Durchschnittlich war das Können eines Absolventen der 10. Schulstufe nötig, um alle 107 untersuchten Anleitungen vollständig verstehen zu können. Dies sei für 80 % aller erwachsenen US-Bürger zu kompliziert, da Bildungsstatistiken von 1993 besagen, dass der durchschnittliche US-Bürger nur auf dem Niveau der 7. Schulstufe liest. Sind Texte zu kompliziert und unverständlich geschrieben, dann gibt ein Mensch für gewöhnlich einfach auf sie weiter zu lesen.

Um gut lesbare, verständliche Texte (Anleitungen) zu schreiben, wurde deshalb das Einhalten folgender Punkte nahe gelegt:

- Verwendung von kurzen, einfachen Wörtern
- Vermeidung von (Fach-)Jargon
- Verwendung einer kultur- und geschlechtsneutralen Sprache
- Einhaltung der korrekten Grammatik, Satzzeichensetzung und Schreibung
- Verwendung von einfachen Sätzen im Aktiv und im Präsens
- Imperativsätze sollen mit einem „action-verb“ begonnen werden
- Verwendung von einfachen graphischen Hilfsmitteln wie Nummerierungen und Listen [11]

Obwohl die Beurteilung von solchen Kriterien eine relativ komplizierte Sache ist, die in den meisten Fällen nur manuell von Experten durchgeführt werden kann, so gibt es doch eine Reihe von Readability-Kennzahlen, die bei der Einstufung von Texten helfen und automatisiert berechnet werden können. In den folgenden Absätzen werden nun einige der bekanntesten Formeln vorgestellt.

4.1 Readability-Kennzahlen

Mit Readability-Formeln wird versucht, die Lesbarkeit von Texten formal zu bestimmen und den Bestimmungsprozess zu automatisieren. Damit sind auch Laien in der Lage, von ihnen verfasste Texte zumindest einigen Voruntersuchungen zu unterziehen, um auf mögliche Lesbarkeitsprobleme aufmerksam zu werden und komplizierte Bereiche einfacher zu formulieren oder Experten hinzuzuziehen. Die Kennzahlen stützen sich dabei in den meisten Fällen auf strukturelle Werte, wie zum Beispiel die durchschnittliche Satzlänge in Wörtern oder die durchschnittliche Buchstaben- oder Silbenanzahl pro Wort. Dies sind auch die beiden Werte, denen in Studien die größte Aussagekraft zugeschrieben wird. Readability-Kennzahlen werden mittels Regressionsanalysen bestimmt, indem die Gleichung gefunden wird, die den Zusammenhang zwischen der empfundenen Schwierigkeit eines Textes (Variable 1) und den oben genannten linguistischen Charakteristiken des Textes (Variable 2) am besten ausdrückt [24].

4.1.1 Flesch-Kincaid Grade Level

Eine der populärsten, verlässlichsten und am meisten getesteten Formeln ist die „Reading Ease"-Formel von Rudolf Flesch. Der gebürtige Österreicher und Jura-Absolvent kam 1938 als Kriegsflüchtling in die Vereinigten Staaten und da sein Juraabschluss dort nicht anerkannt wurde, beschäftigte er sich fortan mit Bibliothekswissenschaften an der Columbia University, wo er 1940 Mitglied der Readability-Forschungsgruppe wurde. Seine in der Folgezeit publizierte Reading Ease Formel zeigte bald, dass Zeitschriften und Verlage eine um 40 bis 60 % größere Leserschaft erreichen konnten, wenn sie Texte verfassten, die gute Werte bei Fleschs Formel erreichen [11]:

$$FRE = 206{,}835 - 1{,}015 \times \left(\frac{\#Wörter}{\#Sätze} \right) - 84{,}6 \times \left(\frac{\#Silben}{\#Wörter} \right)$$

Formel 4.1: Flesch Reading Ease

Die Formel liefert einen Wert zwischen 1 und 100, wobei niedrige Werte für sehr schwere Texte und hohe Werte für sehr einfache Texte stehen (vergleiche Tabelle 4.1 unten).

Reading-Ease Wert (FRE)	Beschreibung	Geschätzte Schulstufe	Geschätzter Anteil der erwachsenen U.S.-Bevölkerung auf diesem Niveau (1949)
0 bis 30	Sehr schwierig	College-Absolventen	4,5 %
30 bis 50	Schwierig	13. – 16.	33 %
50 bis 60	Ziemlich schwierig	10. – 12.	54 %
60 bis 70	Standard	8. – 9.	83 %
70 bis 80	Ziemlich leicht	7.	88 %
80 bis 90	Leicht	6.	91 %
90 bis 100	Sehr leicht	5.	93 %

Tabelle 4.1: Bedeutung und Umwandlung des Reading Ease von Flesch

1976 wurde die Formel im Zuge einer Studie der U.S. Navy modifiziert, um Schulstufen als Ausgabewert zu liefern (U.S. Grade Level):

$$FleschKincaidGradeLevel = 0{,}39 \times \left(\frac{\#Wörter}{\#Sätze} \right) + 11{,}8 \times \left(\frac{\#Silben}{\#Wörter} \right) - 15{,}59$$

Formel 4.2: Flesch-Kincaid Grade Level

Flesch betrachtete also die durchschnittliche Satzlänge und die durchschnittliche Wortlänge (bezüglich der Silbenanzahl) eines Textes und leitete daraus die Schwierigkeit des Textes ab.

4.1.2 Gunning-Fog Index

Robert Gunning gründete 1944 die erste Firma für Readability-Beratung in den USA. Die Firma betreute hauptsächlich Zeitschriften und Nachrichtenmagazine und half dabei mit, deren Readability-Anforderungen von der 16. auf die 11. Schulstufe herabzusetzen. 1952 veröffentlichte Gunning seine „Fog Readability Formula" [12].

$$GunningFogIndex = 0{,}4 \times \left(\left(\frac{\#Wörter}{\#Sätze} \right) + 100 \times \left(\frac{\#komplexeWörter}{\#Wörter} \right) \right)$$

Formel 4.3: Gunning-Fog Index

(#komplexeWörter ... Anzahl der Wörter mit drei oder mehr Silben)

Im Unterschied zu Flesch wird hier nicht mehr die durchschnittliche Wortlänge des Textes betrachtet, sondern das Verhältnis der komplizierten (mehrsilbigen) Wörter zu deren Gesamtzahl. Gunning leitete seine Formel außerdem von Testdaten ab, die 90 %iges Textverständnis voraussetzen, weshalb seine Werte höher ausfallen als bei Flesch. Die Überprüfung des Textverständnisses wird in der Readability-Forschung häufig mit Multiple-Choice- oder Lückentests (Cloze-Tests) durchgeführt. Bei den meisten anderen Readability-Formeln werden dabei bereits 75 % richtige Antworten so gewertet, dass der Text verstanden wurde.

4.1.3 SMOG Readability Formel

"Simple Measure of Gobbledygook", kurz SMOG, ist eine Readability-Formel, die aus dem Jahre 1969 stammt und von dem Psychologen G. Harry McLaughlin entwickelt wurde. „Gobbledygook" ist ein englischer Ausdruck für etwas, das übermäßig kompliziert dargestellt/formuliert wurde, die SMOG Formel soll also eine Vereinfachung und Verbesserung von anderen Lesbarkeitsindizes darstellen. Der Name wurde außerdem in Wertschätzung der Arbeit von Robert Gunning (FOG-Index) gewählt, der als erster die Idee hatte, die Schwierigkeit von Texten über die Anzahl drei- und mehrsilbiger Wörter zu bestimmen.

Bisher vorgestellte Formeln sind nach dem Prinzip $a + b \times Wortlänge + c \times Satzlänge$ aufgebaut. Im Gegensatz dazu meint Laughlin, dass dabei übersehen wurde, dass sich die semantische und die syntaktische Textschwierigkeit gegenseitig beeinflussen. Leichte Unterschiede in Wort- oder Satzlänge in einem schwierigen Textabschnitt drücken nicht denselben Schwierigkeitsunterschied aus wie in einem leichten Textabschnitt. Er schlägt deswegen die allgemeine Form $a + b \times Wortlänge \times Satzlänge$ vor, welche die beiden Einflussgrößen multipliziert.

Laughlins angenäherte Berechnungsvorschrift baut auf der Anzahl der drei- und mehrsilbigen Wörter aus drei Stichproben von je 10 Sätzen (vom Beginn, in der Mitte und vom Ende eines Textes) auf. Aus dieser Anzahl (bzw. aus dem nächstgelegenen idealen Quadrat) zieht man nun die Wurzel und addiert dazu die Konstante 3. Das ergibt dann die Schulstufe, die ein Leser erreicht haben muss, um den vorliegenden Text komplett zu verstehen.

Mit dieser Approximation war es damals (ohne Computerunterstützung) möglich, viel schneller zu Ergebnissen zu kommen, als mit vergleichbaren Readability-Kennzahlen und die Ergebnisse waren sogar genauer. Laughlin hatte festgestellt, dass viele Berechnungen mit anderen Formeln, die oft nur auf Stichproben von knapp 100 Wörtern durchgeführt wurden, zu ungenau waren. Seine Variante mit 30 Sätzen ergab im Schnitt eine Stichprobe von ca. 600 Wörtern und deckte auch noch unterschiedliche Bereiche des Textes ab.

Die exakte und auf Fälle mit 30 und mehr Sätzen generalisierte Formel erreicht eine außergewöhnlich hohe Korrelation von 98,5 % mit den tatsächlichen Schulstufen der Leser, welche die untersuchten Beispieltexte vollständig verstanden hatten [25]:

$$SMOG - GradeLevel = 1{,}0430 \times \sqrt{\#komplexeWörter \times \left(\frac{30}{\#Sätze} \right)} + 3{,}1291$$

Formel 4.4: SMOG Grade Level

(#komplexeWörter ... Anzahl der Wörter mit drei oder mehr Silben)

Der SMOG Index tendiert zu höheren Ergebnissen als alle anderen Readability-Kennzahlen, da die Ergebnisse auf 100 %iges Textverständnis ausgerichtet sind und die Formel darauf normiert wurde [39].

Nachdem sich Herr Laughlin vor langer Zeit aus dem Forschungsbereich der Readability zurückgezogen und sich der klinischen Neuropsychologie verschrieben hatte, war dieser sehr verwundert, dass er über eine halbe Million Suchtreffer erhielt, als er vor kurzem mit Google nach „SMOG Formula“ gesucht hatte. Das große Interesse und die vielen weiterführenden Arbeiten und Untersuchungen haben Laughlin dazu bewogen, sein Interesse für Readability neu aufleben zu lassen. Auf http://wordscount.info/smog.html hat er zusammen mit Alain Trottier einen öffentlichen SMOG Rechner aufgesetzt, der auf einem wörterbuchgestützten Silbenzähler basiert und damit noch genauere Ergebnisse als andere silbenbasierte Formeln liefert. Momentan arbeiten die beiden an einem neuen Verfahren, dem Acceptable Reading Measure (ARM), das die Textschwierigkeit nicht mehr einfach an der Wortlänge festmacht, sondern wo der Bekanntheitsgrad von Wörtern in einer Datenbank gespeichert wird. Diese Datenbank wiederum, soll aus Textkorpora mit Lesematerial unterschiedlicher Schulstufen aufgebaut werden – insgesamt mit mehreren Millionen Wörtern Umfang [25].

4.1.4 Coleman-Liau Index

Als nun vermehrt automatisierte Verfahren zur Readability-Überprüfung eingesetzt wurden, zeigte sich, dass es mit Computern nicht so einfach war, die Silbenzahl von Wörtern verlässlich zu bestimmen. Im Gegensatz dazu ist es sehr einfach die Buchstabenanzahl von Wörtern automatisiert zu zählen.

Eine Readability-Formel, die auf der Buchstabenanzahl anstatt auf Silben basiert, ist der Coleman-Liau Index.

$$ColemanLiau - GradeLevel = 5{,}89 \times \left(\frac{\#Zeichen}{\#Wörter} \right) - 30 \times \left(\frac{\#Sätze}{\#Wörter} \right) - 15{,}8$$

Formel 4.5: Coleman-Liau Index

Grundsätzlich folgt diese Formel also demselben Prinzip wie alle ihre Vorgänger, mit dem Unterschied, dass die schwieriger zu bestimmende Variable der Silben durch die Zeichenanzahl ersetzt wurde [8].

4.1.5 Automated Readability Index (ARI)

Eine weitere, zeichenbasierte Formel stammt aus dem Jahre 1967 und wurde für die United States Army entwickelt, um die Lesbarkeit von technischen Dokumenten und Anleitungen zu beurteilen: Der Automated Readability Index [36].

$$ARI = 4{,}71 \times \left(\frac{\#Zeichen}{\#Wörter} \right) + 0{,}5 \times \left(\frac{\#Wörter}{\#Sätze} \right) - 21{,}43$$

Formel 4.6: Automated Readability Index

Zwar besteht zu der zuvor beschriebenen Formel von Coleman und Liau kein nennenswerter Unterschied, zu Vergleichs- und Orientierungszwecken wird aber trotzdem eine zweite Formel dieser Art in die Umsetzung dieser Arbeit eingebaut werden.

4.1.6 FORCAST

Ein Problem aller bisherigen Formeln ist die Notwendigkeit, dass der zu untersuchende Text als Fließtext in Satzform vorliegen sollte. Dinge wie Listen, Tabellen, Aufzählungen und Überschriften innerhalb des Textes verfälschen das Ergebnis, da sich Kennzahlen wie die Anzahl der Wörter pro Satz ändern. Vor allem für automatisierte Tests mit Computern ist es dann schwierig, nur passende Bereiche auszuwerten, beziehungsweise festzulegen, wo ein Satz endet und ein neuer beginnt. Für die U.S. Army war es wichtig eine Readability-Formel zu haben, die man zum Beispiel auf technische Handbücher anwenden konnte, deren Inhalt häufig nicht in Satzform ist. Genau so eine Formel ist aber auch hilfreich für die Bewertung von Webtexten, wo lange Fließtexte in Satzform auch zugunsten der „Scannbarkeit" von Texten (viele Überschriften, Hervorhebungen, Gliederungen, …) vermieden werden [26].

Die Human Resources Research Organisation entwickelte deshalb im Auftrag der Army eine Formel, die auf die Satzstruktur der Eingabetexte verzichten konnte: Die FORCAST Readability Formel, benannt nach ihren Entwicklern FORd, CAylor und STicht.

$$FORCAST - GradeLevel_{150} = 20{,}43 - 0{,}11 \times (\#EinsilbigeWörter)$$

Formel 4.7: FORCAST Grade Level (150 Wörter)

Die Formel wurde für einen Textausschnitt von 150 Wörtern normiert. Die für längere Texte generalisierte Form lautet wie folgt:

$$FORCAST - GradeLevel = 20{,}43 - 0{,}11 \times \left(\#EinsilbigeWörter \times \frac{150}{\#Wörter} \right)$$

Formel 4.8: FORCAST Grade Level (allgemeine Form)

FORCAST zählt also die Anzahl der einsilbigen Wörter eines Textes und stuft den Text als umso einfacher ein, je größer der Anteil der Einsilber an den Gesamtwörtern ist.

Tabelle 4.2 unten zeigt eine Übersicht, wie die Ergebnisse von FORCAST mit denen von anderen bekannten Readability-Formeln, welche die Satzform verwenden, korrelieren. Der minimale Schulstufenwert (MSW) gibt dabei die kleinste Schulstufe an, bei der noch 50 % der Probanden auf dieser Stufe einen nachher durchgeführten Test zum Textverständnis bestanden haben. Die Kennzahlen wurden auf 12 verschiedene Textabschnitte angewandt und die durchschnittliche Korrelation berechnet:

Kennzahl	Korrelation				Durchschnittliche Schulstufe	Maximale Abweichung
	1	2	3	4		
1 FORCAST	-	0,92	0,94	0,87	10,6	1,9
2 Flesch	0,92	-	0,97	0,92	11,8	4,4
3 Dale-Chall	0,94	0,97	-	0,93	11,6	3,9
4 MSW	0,87	0,92	0,93	-	9,9	2,5

Tabelle 4.2: Korrelation von FORCAST mit anderen Indizes [11]

Weitere Tests mit anderen Texten haben die Gültigkeit von FORCAST im Vergleich zu bisherigen Indizes bestätigt [11].

4.2 Wortschatztests

Eine andere Variante von Readability-Tests sind Verfahren, die neben strukturellen Merkmalen (z. B. Satz- und Wortlänge) auch das verwendete Vokabular, den Wortschatz, betrachten. Zu diesem Zweck verfügen solche Ansätze über Wortschatz-Dateien (z. B. geordnete Listen der häufigsten Wörter einer Sprache/eines Fachbereichs) und prüfen diese auf Überschneidungen mit dem zu untersuchenden Text. Der Bekanntheitsgrad des verwendeten Vokabulars ist ein starker Indikator für das Textverständnis: Die 100 am häufigsten verwendeten Wörter machen im Englischen bereits 25 % der Gesamtwörter von Texten aus und wenn man die 300 häufigsten Wörter betrachtet, entspricht dies bereits einem 65 %igen Anteil [11].

4.2.1 Dale-Chall Readability Formel

Edgar Dale und Jeanne S. Chall waren der Meinung, dass man einen Text in möglichst einfachen und bekannten Wörtern verfassen sollte, um eine hohe Lesbarkeit zu gewährleisten und haben aus diesem Grund eine Liste mit 763 bekannten und einfachen Wörtern erstellt, die 80 % der Schüler der 4. Schulstufe bereits verstehen. 1995 wurde diese Liste überarbeitet und auf 3000 Wörter vergrößert. Diese Liste verwenden sie nun in ihrer Readability-Formel, die auf dem Anteil der unbekannten Wörter (Wörter, die nicht auf der Dale-Chall Liste sind) und der durchschnittlichen Satzlänge basiert [11][33].

$$DaleChall - Score = 0{,}1579 \times (\%SchwierigeWörter) + 0{,}496 \times \left(\frac{\#Wörter}{\#Sätze}\right) + 3{,}6365$$

Formel 4.9: Dale-Chall Score

(%SchwierigeWörter … der Prozentsatz der Wörter, die nicht auf der Dale-Chall Liste der 3000 bekanntesten Wörter gefunden wurden)

Als Ergebnis liefert die Formel einen Wert, der noch auf die exakten U.S. Schulstufen umgerechnet werden muss. Tabelle 4.3 unten veranschaulicht dieses Mapping der Werte.

Dale-Chall Wert	Schulstufe	Bildungsstufe	Alter
4,9 und kleiner	1. bis 4.	Primary	5 – 10
5,0 – 5,9	5. bis 6.		10 – 12
6,0 – 6,9	7. bis 8.	Secondary	12 – 14
7,0 – 7,9	9. bis 10.		14 – 16
8,0 – 8,9	11. bis 12.		16 – 18
9,0 – 9,9	13. bis 15.	College	18 – 22
10 und mehr	16. und höher	University	22+

Tabelle 4.3: Mapping der Dale-Chall Ergebniswerte [33]

Die Ergebnisse von Dale und Chall gelten als noch genauer als der Flesch-Kincaid Grade Level.

4.2.2 The Academic Word List

Bei der Academic Word List (AWL) handelt es sich um eine Liste aus den 570 häufigsten englischen Wortfamilien, die verstärkt in akademischen Texten vorkommen, aber nicht gleichzeitig in den 2000 häufigsten Wörtern der englischen Sprache enthalten sind. Die AWL wurde 2000 an der Victoria University in Wellington, Neuseeland, unter der Leitung von Averil Coxhead entwickelt, um Anhaltspunkte zu bekommen, in welcher Reihenfolge man Vokabeln lernen sollte, um schnellst- und bestmöglich mit dem Lesen von akademischen Texten zurecht zu kommen.

Die in die Liste aufgenommenen Wortfamilien wurden nach folgenden Kriterien ausgewählt:

Zuerst mussten die Wortfamilien in allen drei großen Teilbereichen sowie in mindestens der Hälfte der Fachbereichsuntergruppen des Academic Corpus (siehe Tabelle 4.4 unten) vorkommen. Außerdem mussten die Familien mit einer minimalen Häufigkeit auftreten (mindestens 100 Gesamtvorkommen und mindestens 10 Mal in jedem der drei Teilbereiche). Diese Kriterien sorgen dafür, dass die Wörter für möglichst alle akademischen Fachrichtungen gleichermaßen relevant sind. Nicht berücksichtigt wurden Wörter aus den 2000 häufigsten der englischen Sprache, Wörter, die nur in wenigen Fachbereichen typisch waren (Fachausdrücke), Eigennamen und lateinische Wörter. Ganze Wortfamilien wurden aus dem Grund verwendet, da verschiedene Abwandlungen der Stammform im Allgemeinen auch verstanden werden, sobald die Stammform bekannt ist. Inklusive dieser Abwandlungen hat die AWL einen Umfang von über 3100 Wörtern [9].

Arts	**Commerce**	**Law**	**Science**
Education	Accounting	Constitutional Law	Biology
History	Economics	Criminal Law	Chemistry
Linguistics	Finance	Family Law and Medico-Legal	Computer Science
Philosophy	Industrial Relations	International Law	Geography
Politics	Management	Pure Commercial Law	Geology
Psychology	Marketing	Quasi-Commercial Law	Mathematics
Sociology	Public Policy	Rights and Remedies	Physics

Tabelle 4.4: Zusammensetzung des Academic Corpus [9]

Insgesamt enthielt der Korpus, aus dem die AWL generiert wurde, über 3,5 Millionen laufende Wörter, die in etwa gleichmäßig auf die einzelnen Fachbereiche verteilt waren. Es wurde auch versucht, die Anzahl der Texte möglichst gleichmäßig aus den Sektionen auszuwählen und nur Texte ab einer Länge von 2000 laufenden Wörtern und mehr zu berücksichtigen, da mit längeren Texten verlässlichere Aussagen über die Häufigkeit und das Vorkommen der Wörtern getroffen werden können.

Mit Hilfe der AWL kann man also testen, wie groß die Anteile eines Textes sind, die auf akademischem Niveau formuliert wurden. Weiters wäre es möglich, aus dem Academic Corpus Häufigkeitslisten der einzelnen Fachbereiche zu generieren, die für den Bereich typische Wörter enthalten und mit deren Hilfe man neue, unbekannte Texte und Webseiten eventuell nach Fachgebieten klassifizieren könnte, um Rückschlüsse auf die Readability für bestimmte Zielgruppen zu ziehen. Auf eine Anfrage nach der Verfügbarkeit solcher Listen für die einzelnen Fachbereiche, hat Frau Coxhead aber leider geantwortet, dass die Daten dafür momentan noch nicht vorliegen.

Weiters haben Studien gezeigt, dass das Verstehen und das Lernen neuer Wörter auch unterbewusst beim Lesen, durch das Erfassen von Zusammenhängen und das Verständnis des Gesamtsinns des Texts, möglich ist. Dazu darf der Anteil der unbekannten Wörter im Fließtext allerdings nicht größer als 5 % der Gesamtwörter sein. Anders ausgedrückt, wenn also auf 20 bereits bekannte Wörter nicht mehr als ein neues, unbekanntes trifft, ist es möglich, dass sich dessen Bedeutung aus dem Zusammenhang ergibt und es somit „automatisch“ verstanden wird [7].

4.3 Farben und Kontraste

Farben haben für die menschliche Wahrnehmung eine große Bedeutung. Man sieht nicht nur einen bestimmten Farbton, sondern unterschiedliche Farben rufen auch verschiedenste Empfindungen hervor. So werden zum Beispiel der Farbe Rot Eigenschaften wie Aktivität, Dynamik und Leidenschaft zugeschrieben, während Grün für Entspannung und Ruhe steht oder Blau

Harmonie, Zufriedenheit und Hoffnung ausdrücken kann. Natürlich ist die psychologische Wirkung der Farben sehr individuell geprägt, aber deren grundsätzliche Wirkung wurde vielfach wissenschaftlich bestätigt [23].

Die bewusste Verwendung von Farbe ist deshalb allgegenwärtig (Kunst, Mode, Marketing, …) und natürlich auch im Webdesign ein wichtiger Faktor, um Seiten entsprechend zu präsentieren und zu vermarkten. Es ist jedoch auch Vorsicht geboten, um über die Farbe nicht bestimmte Menschen auszuschließen, weil sie bestimmte Bereiche nicht mehr (oder falsch) wahrnehmen können.

Farben sind Licht mit unterschiedlicher Wellenlänge. Die menschliche Farbwahrnehmung läuft über vier Typen von Rezeptoren auf der Netzhaut des Auges ab: Einmal gibt es dort Stäbchen, die für die Unterscheidung von Hell und Dunkel zuständig sind und dann noch drei Arten von Zapfen, die auf unterschiedliche Farbtöne (Wellenlängen) reagieren. Vereinfacht gesagt, reagieren sie auf Rot, Grün und Blau. Wenn nun bei manchen Menschen bestimmte (oder mehrere) Sorten dieser Zapfen fehlen, dann kommt es zu Farbfehlsichtigkeiten bis hin zur gänzlichen Farbenblindheit.

Ungefähr 8 % der amerikanischen Bevölkerung leidet unter einer bestimmten Form der Farbsehschwäche. Großteils sind die betroffenen Personen männlich (ca. 7 %), das hat genetische Gründe, da bei Frauen oft das zweite X-Chromosom (Männer haben nur eines) einen genetischen Defekt ausgleichen kann. 95 % all dieser Farbsehdefekte sind übrigens eine Form der Rot-Grün-Blindheit (siehe Abbildung 4.1 unten) [16].

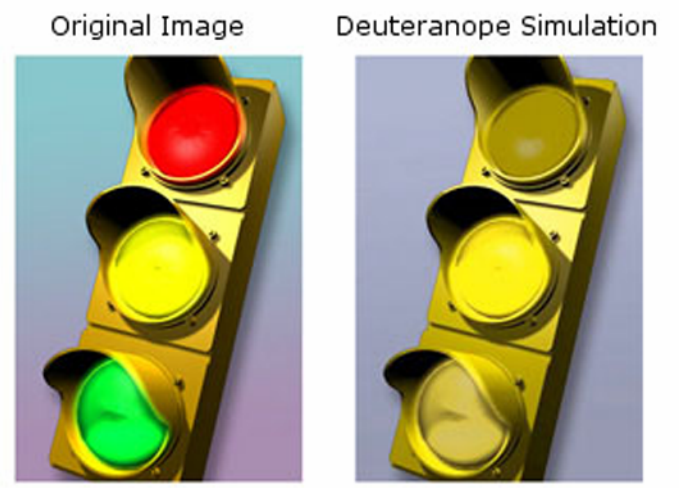


Abbildung 4.1: Simulation einer Rot-Grün-Fehlsichtigkeit

Um diese Bevölkerungsgruppen nicht von Webangeboten auszuschließen, müssen Farben gewählt werden, die auch unterschieden werden können, wenn Menschen mit Farbsehschwächen oder Benutzer mit veralteten Technologien (Schwarz/Weiß-Bildschirm) diese betrachten. Das W3C hat deshalb passend zu der Richtlinie 1.4 der WCAG 2.0 (Unterscheidbarkeit von Vorder- und Hintergrund) ein Erfolgskriterium eingeführt, das ein Helligkeitskontrastverhältnis von mindestens 5:1 zwischen Text und Hintergrund fordert. Dies hilft nicht nur Menschen mit Farbsehschwächen, sondern trägt ganz allgemein zur besseren Erkennbarkeit des Textes bei [58].

$$Helligkeitskontrastverhältnis = \frac{HelligkeitA + 0{,}05}{HelligkeitB + 0{,}05}$$

Formel 4.10: Berechnung des Helligkeitskontrastverhältnisses von zwei Farben (WCAG)

(HelligkeitA … hellerer Text oder Vordergrund, HelligkeitB … dunklerer Text oder Hintergrund)

$$Helligkeit = 0{,}2126 * \left(\frac{Rotwert}{255}\right)^{2{,}2} + 0{,}7152 \times \left(\frac{Grünwert}{255}\right)^{2{,}2} + 0{,}0722 \times \left(\frac{Blauwert}{255}\right)^{2{,}2}$$

Formel 4.11: Berechnung des Helligkeitswertes einer RGB-Farbe (WCAG)

Die Helligkeitswerte können dabei von minimal 0 (Schwarz) bis maximal 1 (Weiß) reichen, was ein maximal mögliches Kontrastverhältnis von 21 bedeutet.

```
Beispiel:

Rot = RGB(255, 0, 0)
Grün = RGB(0, 255, 0)

Helligkeit Rot = 0,2126 + 0 + 0 = 0,2126
Helligkeit Grün = 0 + 0,7152 + 0 = 0,7152

Kontrastverhältnis = 0,7652 / 0,2626 = 2,91
```

Auflistung 4.1: Beispielberechnung des Kontrastverhältnisses von zwei Farben

Ein weiterer Vorschlag, um die Brauchbarkeit des Kontrastes zweier Farben zu berechnen, kommt von der Evaluation and Repair Tools Working Group (ERT-WG) des W3C. Dort wird gesagt, dass zwei Farben eine ausreichende Sichtbarkeit gewährleisten, wenn deren Helligkeitsdifferenz größer als 125 und deren Farbdifferenz größer als 500 ist. Zur Berechnung der entsprechenden Differenzen dienen die folgenden Formeln [48]:

$$Helligkeit = \left(\frac{\left(Rotwert \times 299\right) + \left(GrünWert \times 587\right) + \left(Blauwert \times 114\right)}{1000}\right)$$

Formel 4.12: Berechnung der Helligkeit einer RGB-Farbe (ERT-WG)

$$Helligkeitsdifferenz = abs(HelligkeitA - HelligkeitB)$$

Formel 4.13: Berechnung der Helligkeitsdifferenz von zwei Farben (ERT-WG)

$$\begin{aligned} Farbdifferenz = &\left(\max(Rotwert1, Rotwert2) - \min(Rotwert1, Rotwert2)\right) \\ &+ \left(\max(Grünwert1, Grünwert2) - \min(Grünwert1, Grünwert2)\right) \\ &+ \left(\max(Blauwert1, Blauwert2) - \min(Blauwert1, Blauwert2)\right) \end{aligned}$$

Formel 4.14: Berechnug der Farbdifferenz von zwei RGB-Farben (ERT-WG)

```
Beispiel (wie oben):

Rot = RGB(255, 0, 0)
Grün = RGB(0, 255, 0)

Helligkeit Rot = 255 * 299 / 1000 = 76,25
Helligkeit Grün = 255 * 587 / 1000 = 149,69

Helligkeitsdifferenz = 149,69 - 76,25 = 73,44

Farbdifferenz = 255 + 255 + 0 = 500
```

Auflistung 4.2: Beispielberechnung der Helligkeits- und Farbdifferenz von zwei Farben

4.4 Readability-Evaluatoren

Hilfsmittel zur Beurteilung der Lesbarkeit sind bisher kaum in Accessibility-Tools vorhanden (in keinem der Tools die in Kapitel 3 betrachtet wurden). Es gibt aber eine Reihe von eigenständigen Werkzeugen, die im Web verfügbar sind und zur automatischen Readability-Analyse eingesetzt werden können. Nachfolgend werden zwei davon kurz betrachtet.

4.4.1 TxReadability

Das Readability Tool TxReadability (http://www.lib.utexas.edu:8080/TxReadability/app) wird von dem Accessibility-Institut der Universität Texas in Austin entwickelt. Momentan bietet es Analysemöglichkeiten für die Sprachen Englisch, Spanisch und Japanisch an und es gilt als Pilotprojekt für die Integration von mehreren Sprachen in einem Readability-Analysewerkzeug.

TxReadability baut auf der Verwendung von Readability-Formeln auf, wovon einige bekannte bereits zuvor in Kapitel 4.1 eingeführt wurden. Die Entwickler des Tools haben bei der Auswahl der Formeln auf sechs Kriterien Wert gelegt:

1. Die Anwendbarkeit der Formel auf Webinhalte muss gewährleistet sein.
2. Es müssen passende Algorithmen realisierbar sein, um alle Komponenten der Formel automatisch errechnen zu können.
3. Die Ergebnisse müssen auf Schulstufen übertragbar sein.
4. Die Formel soll diese Ergebnisse für möglichst breit gefächerte Schulstufen liefern können und nicht nur auf bestimmte Stufen spezialisiert sein.
5. Die Formel sollte bereits weitläufig verwendet werden und anerkannt sein.
6. Außerdem sollte sie einer breiten Öffentlichkeit bekannt sein.

Dabei ist man während der Entwicklung von TxReadability zu dem Schluss gekommen, dass Methoden, die Wortlisten zur Analyse der Lesbarkeit verwenden nicht so gut geeignet wären, da zum Einen nur sehr wenige solcher Listen verfügbar sind und diese weiters nicht auf einem aktuellen Stand mit dem im Web vorherrschenden Vokabular sind. Zudem wurde bemerkt, dass die meisten Formeln hauptsächlich auf Kennzahlen beruhen, die ganze Sätze benötigen und dass nur der FORCAST Grade Level eine Berechnungsmethode bietet, die nicht auf diese Satzstruktur angewiesen ist.

Großer Aufwand musste natürlich für die Behandlung der drei unterschiedlichen Sprachen betrieben werden. Man benötigte Algorithmen, um Parameter wie Silben, Satz- oder Worttrennzeichen in den unterschiedlichen Sprachen/Zeichensätzen zu erkennen und korrekt zu erfassen [45].

4.4.2 Readability.info

Ein weiteres frei zugängliches Onlinetool zur Readability-Analyse findet man auf www.readability.info. Nach der Eingabe einer URL oder dem optionalen Upload eines Word-Dokuments, wird ein Readability-Bericht generiert, in dem die Ergebnisse mehrerer bekannter Kennzahlen aufgeführt werden.

Außerdem werden einige Zusatzinformationen zum analysierten Text ausgegeben, wie zum Beispiel die Anzahl der Buchstaben, der Anteil der langen oder kurzen Sätze und die Anzahl an Frage- oder Passivsätzen.

Interessant ist eine Seite mit Beispielergebnissen von bekannten Webseiten. Diese unterstützt Benutzer des Werkzeugs beim Einschätzen der Ausgabewerte der eigenen Seitenanalyse.

Tabelle 4.5 unten zeigt einen Auszug der dort angeführten Ergebnisse von Readability-Formeln, die auch in dieser Arbeit bereits besprochen wurden. Man sieht sofort, dass beispielsweise die Pressemitteilung des Weißen Hauses auffallend niedrige Readability-Anforderungen hat. Offensichtlich ist es der U.S. Regierung wichtig, dass mit ihren Aussendungen möglichst breite Schichten der Bevölkerung erreicht werden.

NY Times: Artikel Flesch-Kincaid: 6,2 ARI: 6,2 Coleman-Liau: 11,8 Fog-Index: 8,9 SMOG: 9,1	**PC World, Artikel** Flesch-Kincaid: 10,6 ARI: 12,1 Coleman-Liau: 11,9 Fog-Index: 13,3 SMOG: 11,4	**Weißes Haus, Pressemitteilung** Flesch-Kincaid: 4,1 ARI: 3,7 Coleman-Liau: 9,1 Fog-Index: 7,6 SMOG: 8,2
Intuitive.com, Homepage Flesch-Kincaid: 9,7 ARI: 10,9 Coleman-Liau: 11,0 Fog-Index: 12,8 SMOG: 11,2	**ESPN.Go.com, Artikel** Flesch-Kincaid: 12,3 ARI: 6,7 Coleman-Liau: 15,6 Fog-Index: 15,7 SMOG: 10,7	**Nickelodean, Homepage** Flesch-Kincaid: 4,0 ARI: 3,5 Coleman-Liau: 9,1 Fog-Index: 6,2 SMOG: 7,0

Tabelle 4.5: Beispielergebnisse von Readability-Formeln [42]

4.5 Diskussion und Schlussfolgerung

Die vorgestellten Readability-Kennzahlen benötigen fast alle Texte, die in Satzform vorliegen. Deshalb ist eine Anwendung auf Webtexte nur begrenzt oder mit Abstrichen möglich. Ein weiteres Problem, das sich bei einer automatisierten Verwendung solcher Kennzahlen ergibt, ist der Umstand, dass Wortsilben von einem Computer nicht zu 100 % genau gezählt werden können. In Summe kann dies zu einer nicht unwesentlichen Verfälschung der Ergebnisse solcher Formeln führen.

Eine Möglichkeit diesen Problemen entgegenzuwirken, ist die Auswahl von Formeln, die auf Zeichen statt auf Silben basieren. Trotzdem bleibt der Nachteil, dass - auch bei den gefundenen Formeln in diesem Bereich – Texte in Satzform nötig sind. Die einzige Formel, die darauf verzichten kann, ist FORCAST. Diese ist allerdings wieder silbenbasiert und wurde außerdem für die U.S. Army entwickelt. Kritische Stimmen meinen, dass sie deshalb hauptsächlich für technische Texte und männliche Testpersonen brauchbare Ergebnisse liefert.

Bis also spezielle Formeln für die Webstruktur von Texten verfügbar sind, muss versucht werden, die Auswahl der Webtexte den Formeln „anzupassen“.

Die Entwickler von FORCAST haben in ihrer Analyse außerdem festgehalten, dass Ansätze, die das verwendete Vokabular einer Seite untersuchen auch nicht auf spezielles, im Web gebräuchliches Vokabular eingehen. Trotzdem kann man durch Vergleiche mit den häufigsten Wörtern einer Sprache sicherlich Rückschlüsse auf die allgemeine Textschwierigkeit ziehen. Auch eine Kombination mit den herkömmlichen Readability-Kennzahlen macht Sinn, da man dadurch die Unzulänglichkeiten der jeweils anderen Formel ein wenig abfangen kann und die strukturelle Untersuchung mit der Betrachtung des Inhalts vereint.

Bei der Analyse der Readability-Evaluatoren fiel dann auf, dass diese häufig Ungenauigkeiten beim Parsen der Quelltexte haben. Es werden oft Bereiche der Seite in die Readability-Überprüfung mit einbezogen, die dort nichts zu suchen haben. Das wären zum Beispiel Passagen, die Code enthalten. Diese Unzulänglichkeiten, die beim Testen der Readability-Evaluatoren mit zufälligen Webseiten aufgedeckt wurden, werden wir in der Umsetzungsphase dieser Arbeit versuchen zu verbessern.

5 Umsetzung

Im 5. Kapitel werden einleitend die Betrachtungen der vorangegangen Kapitel herangezogen, um Bereiche und Möglichkeiten zu identifizieren, wo vorhandene Accessibility-Evaluatoren verbessert werden könnten und wie dies konkret umgesetzt werden kann. Danach wird der Prototyp eines modellbasierten Evaluators vorgestellt, auf den die vorliegende Arbeit aufbauen wird. Es werden das Framework, die Architektur, sowie Erweiterungen und deren grundsätzliche Funktionsweise betrachtet.

5.1 Bereiche mit Verbesserungspotential

In Anlehnung an die Diskussionen und Schlussfolgerungen der vorigen Abschnitte, werden nun Bereiche vorgestellt, die in der Umsetzung des Evaluators dieser Arbeit prototypisch verbessert werden sollen.

5.1.1 Kontextabhängige Betrachtung von Inhalten

Die Prüfung, ob brauchbare Alternativtexte zu multimedialen Inhalten verfügbar sind, ist einer der häufigsten Tests im Accessibility-Bereich. Dies hat den Grund, dass in Schriftzeichen codierte Informationen, im Gegensatz zu Bildern und Tönen, besser von assistiven Technologien erfasst und umgewandelt werden können. Zum Beispiel über die Darstellung auf einem Braille-Blindenschrift-Display (siehe Abbildung 5.1 unten) oder ganz einfach durch maschinelles Vorlesen des Alternativtextes (Screenreader).

Abbildung 5.1: Braille Blindenschrift-Display

Das W3C schreibt hier die generelle Verwendung von Alternativtexten zu allen Nicht-Text-Inhalten, wie zum Beispiel Bildern, vor (siehe Richtlinie 1 der WCAG 1.0, bzw. Prinzip 1 in den neuen WCAG 2.0). Allerdings ist Bild nicht gleich Bild und Alternativtext nicht gleich Alternativtext. Man stelle sich beispielsweise eine optisch ansprechend gestaltete Webseite mit vielen kleinen Grafiken (Ränder, Verzierungen, Gliederungszeichen, ...) vor, die einzig und allein dazu dienen, das Layout zu verwirklichen, aber nichts mit einem konkreten Inhalt zu tun haben. Korrekterweise müssten solche Effekte mit Hintergrundgrafiken (style="background: …", background="…") realisiert werden, die dann auch keine Auswirkungen auf alternative Darstellungsformen haben, da es sich ja um Hintergrundinformationen handelt. In der Praxis werden solche Formatierungseffekte aber trotzdem häufig über gewöhnlich eingebundene Grafiken (<img src="grafik.gif" alt="Eine Grafik">) realisiert, da man dadurch zum Beispiel gleichzeitig eine automatische Größenanpassung von Tabellenzellen erreicht. Solche, oder andere für den Inhalt irrelevante Grafiken, dürfen nun konsequenterweise keinen Alternativtext haben, da diese sonst von Screenreadern mit vorgelesen würden. Kein Alternativtext ist wiederum nicht WCAG-konform und wird von Prüfsoftware als Fehler markiert. Für solche Grafiken wird deshalb ein leerer Alternativtext (alt="") vorgeschlagen, der von assistiven Technologien dementsprechend richtig interpretiert wird.

Das Dilemma für Accessibility-Prüfprogramme ist nun aber, dass sich die korrekte Beurteilung der Alternativtexte weiter erschwert hat, da die Programme nicht unterscheiden (können), ob ein Bild relevant für den Inhalt ist oder nicht. Dies führt zu weiteren, maschinell unentscheidbaren Warnungen, die manuell nachgeprüft werden müssen.

Ein erster Verbesserungsschritt in einem Accessibility-Evaluator wäre also eine vorherige Klassifizierung der Bilder in die Klassen „rein dekorativ" und „inhaltlich relevant". Die so als rein dekorativ identifizierten Bilder können dann eindeutig von dem Tool beurteilt werden, da für sie nur noch der explizit leere Alternativtext in Frage kommt. Gleichzeitig kann man den explizit leeren ALT-Text für alle Bilder, die nicht als rein dekorativ festgelegt wurden, eindeutig als fehlerhaften ALT-Text einordnen. Bei der relativ hohen Anzahl an Grafiken, die heutzutage auf Webseiten verwendet werden, kann das durchaus zu einer spürbaren Reduzierung von manuellen Nachprüfungen führen.

Die eigentliche Klassifizierung der Bilder könnte sich dabei auf drei Arten realisieren lassen:

- Extrahierung der Klasseninformation aus dem Quellcode: Bilder, die in bestimmten Regionen der Webseite vorkommen, könnten von inhaltlicher Relevanz für die vorliegende Webseite ausgeschlossen werden (Werbung?). Für Bilder, die eine bestimmte Größe nicht überschreiten und zum Beispiel zu Größenanpassungszwecken von Tabellen verwendet werden, gilt dies ebenso.
- Einführung eines Zwischenschritts, in dem der Benutzer die Möglichkeit erhält die Klasse von ausgewählten Bildern festzulegen oder wo vom Programm vorgenommene Klassifikationen korrigiert werden können.
- Weiterverarbeitung von Informationen aus vorgelagerten Entwicklungsschritten: So könnten Web-Autorenwerkzeuge schon bei der Erstellung von Inhalten nach dem Zweck einer Grafik fragen und diese Information dann in Form von Metainformationen mit in den Quelltext der Webseite codieren. Prüfprogramme können diese Informationen dann später aufgreifen, um exaktere Analysen durchzuführen.

Alle drei Arten der Informationsgewinnung werden später prototypisch gezeigt und implementiert (vergleiche Kapitel 6).

Eine ähnliche Verbesserungsmöglichkeit wäre auch über die Klassifizierung von Tabellen (Layout- oder Datentabelle) und der kontextabhängigen Ausführung der darauf aufbauenden

Tests (Textzusammenfassungen für Tabelleninhalte, korrekte Verwendung von <th>-Tags, usw.) gegeben.

5.1.2 Integration von Entscheidungshilfen zur Readability

Ein weiterer, schwer zu beurteilender Punkt, der in aktueller Prüfsoftware kaum oder gar keine Beachtung findet, betrifft die Verwendung einer einfachen, leicht verständlichen Sprache. In beiden Versionen der WCAG (Richtlinie 14 bzw. Prinzip 3) wird die Umsetzung dieses Kriteriums mit Priorität 1 gefordert und Accessibility-Programme berücksichtigen dies zumeist mit einer Warnung und der Notwendigkeit einer manuellen Nachprüfung. Es ist wohl ein Faktum, dass eine exakte Prüfung, ob ein Text für eine bestimmte Zielgruppe geeignet ist oder nicht, nur durch einen menschlichen Gutachter verlässlich durchgeführt werden kann. Und selbst so und mit Hilfe von Experten, ist dies kein einfaches Unterfangen, da die Textwahrnehmung und das Textverständnis sehr komplexe und vor allem auch individuelle Prozesse sind. Trotzdem haben jahrelange Forschung und Studien auf diesem Gebiet gezeigt, dass Readability durchaus in sehr hohem Maße mit den strukturellen Kennzahlen von Texten korreliert. Die grundsätzliche und leicht nachvollziehbare Erkenntnis dabei besagt schlicht und einfach, dass ein Text um so verständlicher ist, je einfacher die verwendeten Wort- und Satzkonstruktionen sind (Wortlänge, Silbenanzahl, Satzlänge) und je geläufiger das verwendete Vokabular ist.

Genau in diesen Bereichen kann eine automatisierte Überprüfung aber sehr wohl Anregungen liefern oder mögliche Problembereiche aufzeigen. Natürlich kann man Texte schreiben, die absolut sinnlos und unverständlich sind und trotzdem gute Ergebnisse bei solchen Readability-Formeln erreichen oder umgekehrt. Aber das kann man mit jeder Formel machen deren Eingabeparameter man kennt, wenn man diese absichtlich entsprechend manipuliert. Fakt ist, dass diese Kennzahlen aus tatsächlich und ernsthaft von Menschen verfassten Texten generiert wurden und auch entsprechend getestet und validiert worden sind. Es ist also sicherlich anzustreben, einen Benutzer eines Accessibility-Evaluators mit Hilfe solcher Informationen auf mögliche Problembereiche oder Fehler hinzuweisen, anstatt nur einen manuellen Checkpunkt auszugeben.

Deswegen wird während dieser Arbeit versucht werden, solche Readability-Kennzahlen gemeinsam mit Analysen, die sich mit dem auf der Webseite verwendeten Wortschatz befassen, in einen Evaluator zu integrieren, um Benutzer bei der manuellen Beurteilung dieser Checkpunkte zu unterstützen.

5.1.3 Exakte Berechnung von CSS-Farbwerten und Kontrasten

In keinem der in Kapitel 3 betrachteten Freeware-Accessibility-Programme wurden die Farbgebung und das Kontrastverhältnis von Textpassagen und Hintergrund näher analysiert. Zwar wurden auch hier zumeist allgemeine Warnhinweise ausgegeben, allerdings ist nirgends versucht worden, den tatsächlichen Kontrast von verdächtigen Bereichen zu berechnen und nur an den richtigen Stellen konkrete Fehlermeldungen anstatt allgemeingültiger Warnungen auszugeben. Dazu wurden alle Evaluatoren mit einer Dummy-Webseite getestet, die über Style-Informationen mit extrem unleserlichen Kontrasten formatiert wurde. Bei einigen Prüfprogrammen fehlte sogar der obligatorische Warnhinweis auf ein mögliches Problem. Ob diese dürftige Kontrastanalyse nun damit zusammenhängt, dass Autoren von Prüfprogrammen annehmen, dass man Kontrastverhältnisse schon während des Verfassens von Webseiten selbstständig als nicht ausreichend identifizieren würde, oder ob es nicht so einfach ist, aus dem HTML-Code der Seite auf die genaue Farbgebung der einzelnen Teilbereiche zu schließen, kann nur vermutet werden. Jedenfalls ist es sicherlich nicht ausreichend, es dabei zu belassen, dass ein Webautor dies im wahrsten Sinne des Wortes mit „Augenmaß“ beurteilen muss. Schlechte Kontraste wirken sich nämlich zuerst bei Menschen mit Sehschwächen aus, wenn man als normalsichtiger Webautor noch gar keine Schwierigkeiten wahrnimmt.

In den WCAG (1.0: Richtlinie 2, 2.0: Prinzip 1) wird durch verschiedene Checkpunkte allerdings ganz klar die Notwendigkeit für ausreichende Kontraste formuliert und durch entsprechende Formeln auch berechenbar gemacht (vergleiche Kapitel …). Es handelt sich hierbei also um einen eindeutig verbesserungswürdigen Punkt, der auch umsetzbar ist und in Folge, in das hier entwickelte Accessibility-Tool, eingebaut werden soll.

5.1.4 Visualisierung von strukturellen Problemen

Viele Kriterien in den Accessibility-Regelwerken haben sich mit der Strukturierung von Webseiten beschäftigt. Strukturierung in dem Sinne, dass eine Webseite so aufgebaut werden soll, dass deren Grundgerüst auch noch funktioniert und zugänglich ist, wenn sämtliche Zusatztechnologien deaktiviert oder nicht verfügbar sind oder gar nicht wahrgenommen werden können (Richtlinie 2 und 6 beziehungsweise Prinzip 4 in den WCAG). In den Guidelines wird dies auch als „geschmeidige Transformation“ der Webseite bezeichnet.

Gemeint ist damit zum Beispiel, dass Informationen nicht nur ausschließlich über eine unterschiedliche Farbgebung dargestellt werden dürfen, da farbenblinde Benutzer oder Schwarz/Weiß-Monitore diese Information nicht wahrnehmen bzw. darstellen können. Die entsprechenden Bereiche müssen also auch im Markup so strukturiert sein, dass sie unterschieden werden können. Das selbe gilt für Seiten die optionale Technologien wie Cascading Stylesheets (CSS) oder Scriptsprachen einsetzen. Auch wenn diese Zusätze nicht verfügbar sind (veralteter Browser oder vom Benutzer deaktiviert), sollte die Grundfunktionalität der Seite erhalten bleiben.

In den betrachteten Evaluatoren wurden diese Richtlinien nur sehr oberflächlich betrachtet. Meistens wird einfach der Wortlaut der Richtlinie in Form einer Warnung wiedergegeben, die vom Benutzer überprüft werden soll. Natürlich handelt es sich hier um Fälle, die so gut wie nicht automatisiert bearbeitet werden können, aber aktuelle Prüfprogramme bieten den Nutzern auch wenige bis gar keine Hilfen an, die manuelle Überprüfung zumindest zu vereinfachen und zu beschleunigen. Im erweiterten Prototyp soll später gezeigt werden, dass es mit relativ einfachen Mitteln aber doch möglich ist, solche Hilfsmittel anzubieten. Diese könnten versuchen, die Webseite so darzustellen, wie sie für Benutzer mit Einschränkungen oder veralteten Technologien aussehen würde. Mit solchen Visualisierungen sollte es möglich sein, Probleme in diesen Bereichen besser und schneller zu identifizieren, zu verstehen und zu beheben.

5.2 Der Prototyp: WUSAB

Die Umsetzung dieser Arbeit wird auf dem Prototyp eines Usability-Evaluators basieren, der von Richard Atterer, einem Doktoranden an der Lehr- und Forschungseinheit Medieninformatik der LMU München, entwickelt wurde. Der Prototyp, WUSAB (WebUSABility), greift die Idee auf, dass während der Evaluierung von Webseiten zusätzlich auf Modelle geachtet werden muss, welche die Webseite beziehungsweise deren Bereiche und Inhalte näher beschreiben. Dadurch sollen genauere und bessere Analysen möglich werden, die dazu beitragen, die Usability der Webseite weiter zu verbessern [2].

5.2.1 Framework

Bei dem Prototyp handelt es sich um ein serverseitiges Programm mit einem Webinterface. Umgesetzt wurde dies auf einem Apache Tomcat Application Server mit der JavaServer Pages/Servlet Technologie.

Apache Tomcat

Der Tomcat Application Server ermöglicht die Ausführung von Java-Code auf Webservern. Dies geschieht in dem so genannten Servlet-Container. Tomcat inkludiert außerdem einen eigenständigen Webserver und bietet damit eine in sich geschlossene Plattform, um dynamische Webapplikationen mit Java zu erstellen [1].

Java Servlets

Ein Servlet ist ein Java-Programm, das im Servlet-Container des Application Servers ausgeführt wird und Anfragen von Clients entgegennehmen und dynamisch beantworten kann. Damit ist es also möglich, die normale Funktionalität eines Webservers mit eigener Logik zu erweitern [40].

JavaServer Pages (JSP)

JavaServer Pages sind eine Erweiterung der Servlet-Technologie. Ähnlich wie mit PHP kann man mit JSP dynamische Inhalte in statisches HTML (oder XML) einbinden. Dies kann über vordefinierte JSP-Aktionen oder durch das Einfügen von Java-Quellcode in die HTML-Seite erfolgen. JavaServer Pages können wie Servlets auf einem Tomcat-Server ausgeführt werden, dazu werden sie mit dem JSP-Compiler Jasper in ein entsprechendes Servlet umgewandelt, welches dann im Servlet-Container ausgeführt werden kann.

Mit JSP ist es möglich, das Model-View-Controller Pattern (MVC) umzusetzen und dadurch eine Trennung von Anwendungslogik und Präsentation zu erreichen. Dies hat den Vorteil, dass man einzelne Teile der Anwendung austauschen und verändern kann, ohne dass der Rest dadurch beeinträchtigt wird, solange die selben Schnittstellen verwendet werden. Abbildung Abbildung 5.2 unten zeigt eine schematische Darstellung des MVC-Pattern, die View wird dabei durch JSP-Seiten realisiert [40].

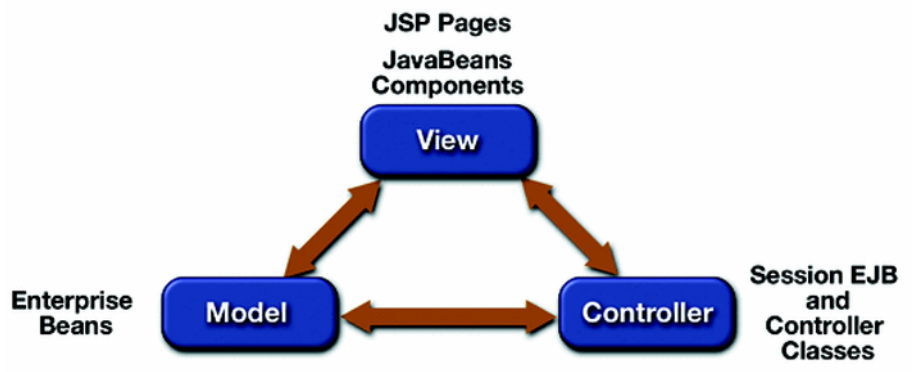


Abbildung 5.2: Model-View-Controller Pattern

5.2.2 Architektur

Der ursprüngliche Prototyp besteht aus der eigentlichen Anwendung im main-Paket, einem externen HtmlParser und einigen JSP-Dateien, die für die dynamische Generierung des Webinterfaces zuständig sind.

Abbildung 5.3 unten zeigt das Klassendiagramm des Prototyps. Die Klasse PageSession stellt darin ein zentrales Element dar. Sie wird direkt nach dem Übermitteln einer URL an den Evaluator instanziiert (PageSession.create(session, url)) und speichert alle Daten der aktuellen Sitzung. Zum Beispiel hält sie eine Referenz auf das Document Object Model (DOM) der zu prüfenden Webseite in der Instanzvariable „document“. Dieses Modell wird nach dem Download der Zielwebseite von dem HtmlParser erstellt und als eine Liste von Elementen in einem

HtmlDocument-Objekt zurückgeliefert. Die mit JSP-Dateien realisierte Schnittstelle zum Benutzer (View), gelangt über einen Aufruf der Methode PageSession.get(session, url) im Kopf jeder Seite an die Daten der aktuellen Sitzung und kann dann mit diesen weiterarbeiten. Dies geschieht unter anderem durch das Einbinden verschiedener HtmlVisitor-Objekte, mit deren Hilfe das Dokument durchlaufen, manipuliert und entsprechend ausgegeben werden kann. Ein spezieller HtmlVisitor ist der RegionSelDumper, der den HTML-Quellcode mit kleinen verlinkten Icons annotiert, die verwendet werden, um bestimmte Bereiche der Seite auszuwählen und ihnen Eigenschaften zuzuweisen. Für diese Bereiche werden vom RegionSelDumper PageArea-Objekte angelegt, welche die Daten der einzelnen Seitenbereiche speichern und wiederum gesammelt in einem Vektor in der PageSession abgelegt werden.

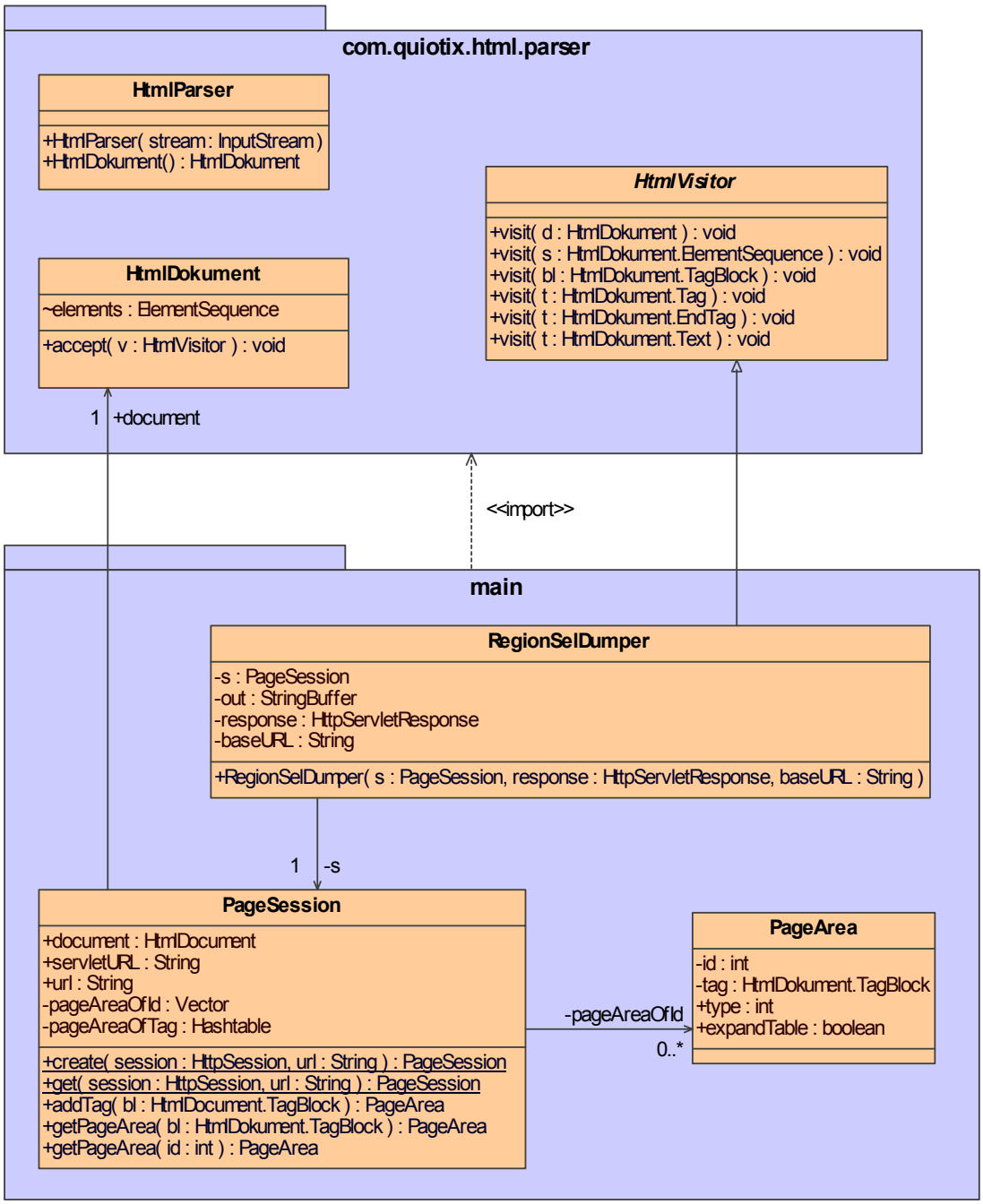


Abbildung 5.3: Klassendiagramm des ursprünglichen Prototyps

5.2.3 Funktionsweise

Im Gegensatz zu bisherigen Usability- bzw. Accessibility-Evaluatoren führt der Prototyp von Herrn Atterer einen Zwischenschritt in den Evaluierungsprozess ein, der vom Benutzer dazu verwendet werden kann, den einzelnen Bereichen der analysierten Seite ihre Funktionalität zuzuordnen (vergleiche Abbildung 5.4 und Abbildung 5.5). Die folgenden Typen können vergeben werden:

- Logo
- Werbung
- Navigation
- Hauptinhalt
- Zusatzinhalt
- Suchformular
- kein spezieller Bereich

Abbildung 5.4: WUSAB - Auswahl einer Seitenregion

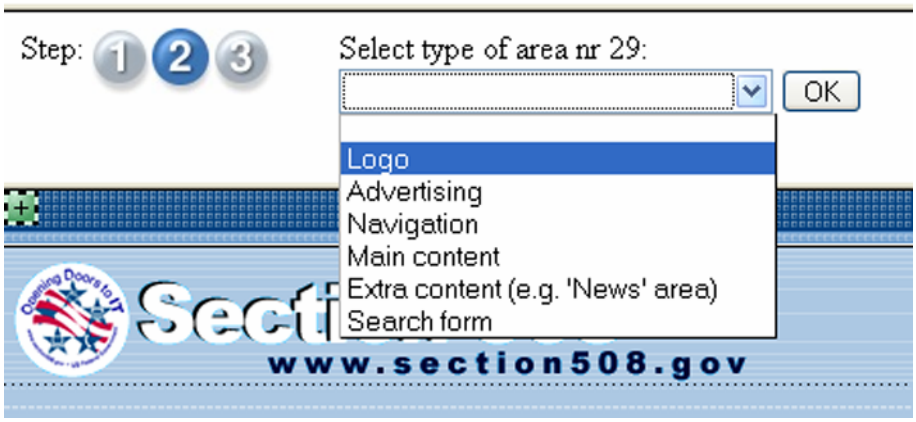


Abbildung 5.5: WUSAB - Zuweisung eines Typs zu einer Seitenregion

Wenn der Benutzer dann alle wichtigen Teile der Webseite typisiert hat, startet er die eigentliche Auswertung, wo diese Zusatzinformationen nun zu detaillierteren Analysen herangezogen werden.

5.3 Der erweiterte Prototyp

Um die unter Punkt 5.1 genannten Verbesserungsmöglichkeiten in den Evaluator zu integrieren, sind einige Erweiterungen und Umstrukturierungen nötig. Allem voran wird für einige Anwendungen (z. B. bei einer Analyse der Farbkontraste) neben dem Dokumentmodell der HTML-Seite auch ein Modell der Stylesheet-Informationen (CSS) benötigt.

5.3.1 Architektur

Der neue Aufbau der Anwendung berücksichtigt zusätzlich zu den bestehenden Pakten der Hauptanwendung (main) und des HtmlParsers (com.quiotix.html.parser) also noch einen CSS-Parser sowie eine Hilfsanwendung, um HTML-Code umzuwandeln (z. B. Konvertieren von HTML-Entities). Abbildung 5.6 unten veranschaulicht die Paket-Architektur. Als CSS-Parser wurde ein Projekt von David Schweinsberg (http://cssparser.sourceforge.net/) importiert, das einen Document Object Model (DOM) Level 2 Style-Baum aus den geparsten CSS-Informationen generiert. Dazu implementiert die Parseranwendung (com.steadystate.css.*) ein vom W3C zur Verfügung gestelltes Interface-Paket für das CSS-DOM.

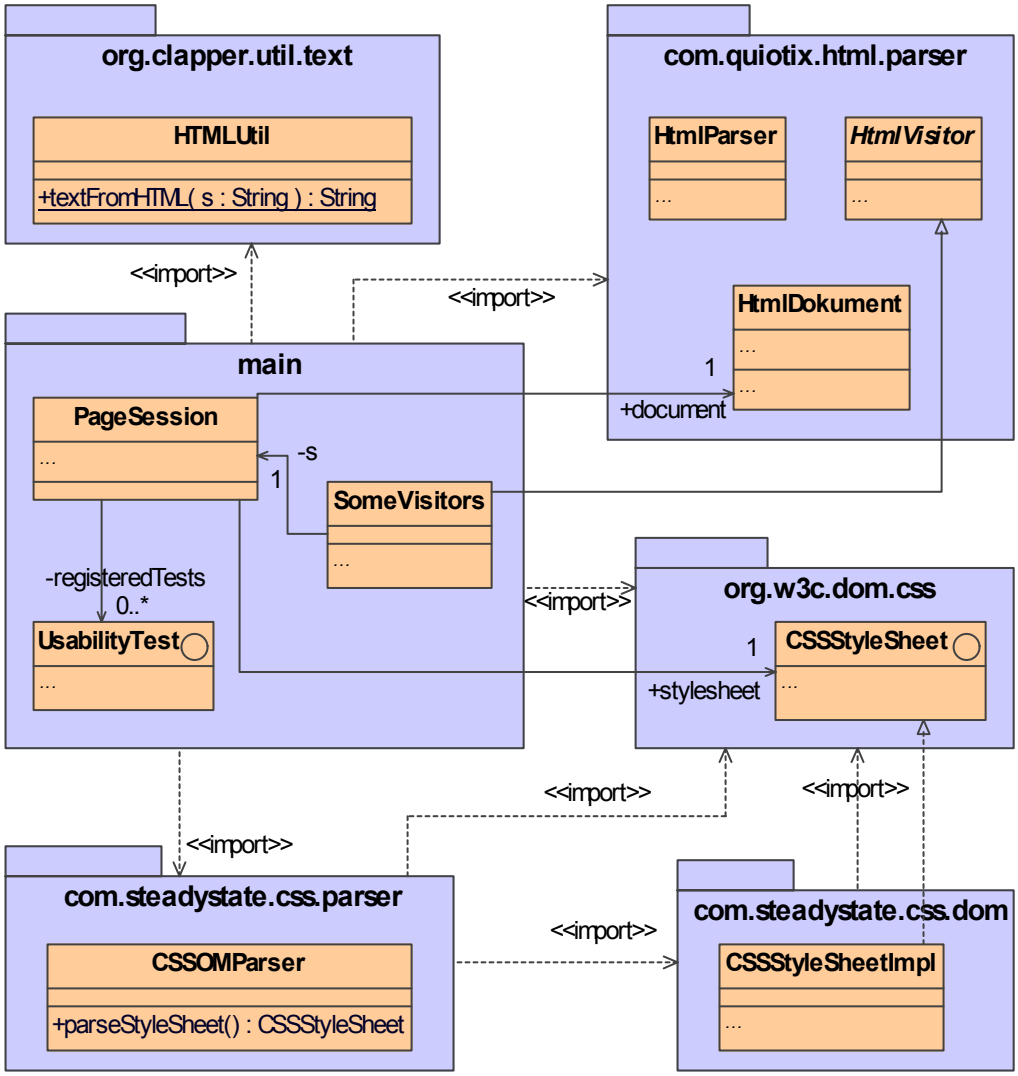


Abbildung 5.6: Pakete des erweiterten Prototyps

Die Hauptanwendung befindet sich nach wie vor im Paket „main“ (siehe Abbildung 5.7 unten) und wurde um entsprechende Funktionalität für die umzusetzenden Accessibility-Tests erweitert.

Zuerst wurde mit dem Interface UsabilityTest eine gemeinsame Schnittstelle für alle zukünftigen Testfälle geschaffen. In dem Vektor registeredTests in der PageSession können nun Objekte vom Typ UsabilityTest referenziert und dadurch für eine spätere Ausführung während der Analysephase registriert werden. Das Interface fordert von allen implementierenden Klassen Methoden, um die Testausführung zu steuern und die Ergebnisse der Analysen abzufragen und

regelt dadurch die einheitliche Handhabung der Tests. Geplant sind Analysen der Lesbarkeit, der Farbkontraste, der Seitenstruktur und der Alternativtexte von Bildern. Die detaillierte Implementierung dieser Testfälle und damit zusammenhängender Mechanismen werden in Kapitel 6 beschrieben.

Für die Klasse PageArea aus dem ursprünglichen Prototyp wurde eine Oberklasse PageElement eingeführt, die gemeinsame Attribute aller Typen von Seitenelementen enthält. Für den oben bereits erwähnten Test der Alternativtexte wird eine zusätzliche Unterklasse PageImage eingeführt. Diese enthält spezifische Informationen zu der Grafik, wie zum Beispiel die Referenz auf den <img> Tag im Dokumentbaum, Größen- und Positionsangaben und das Ergebnis der Bildklassifizierung.

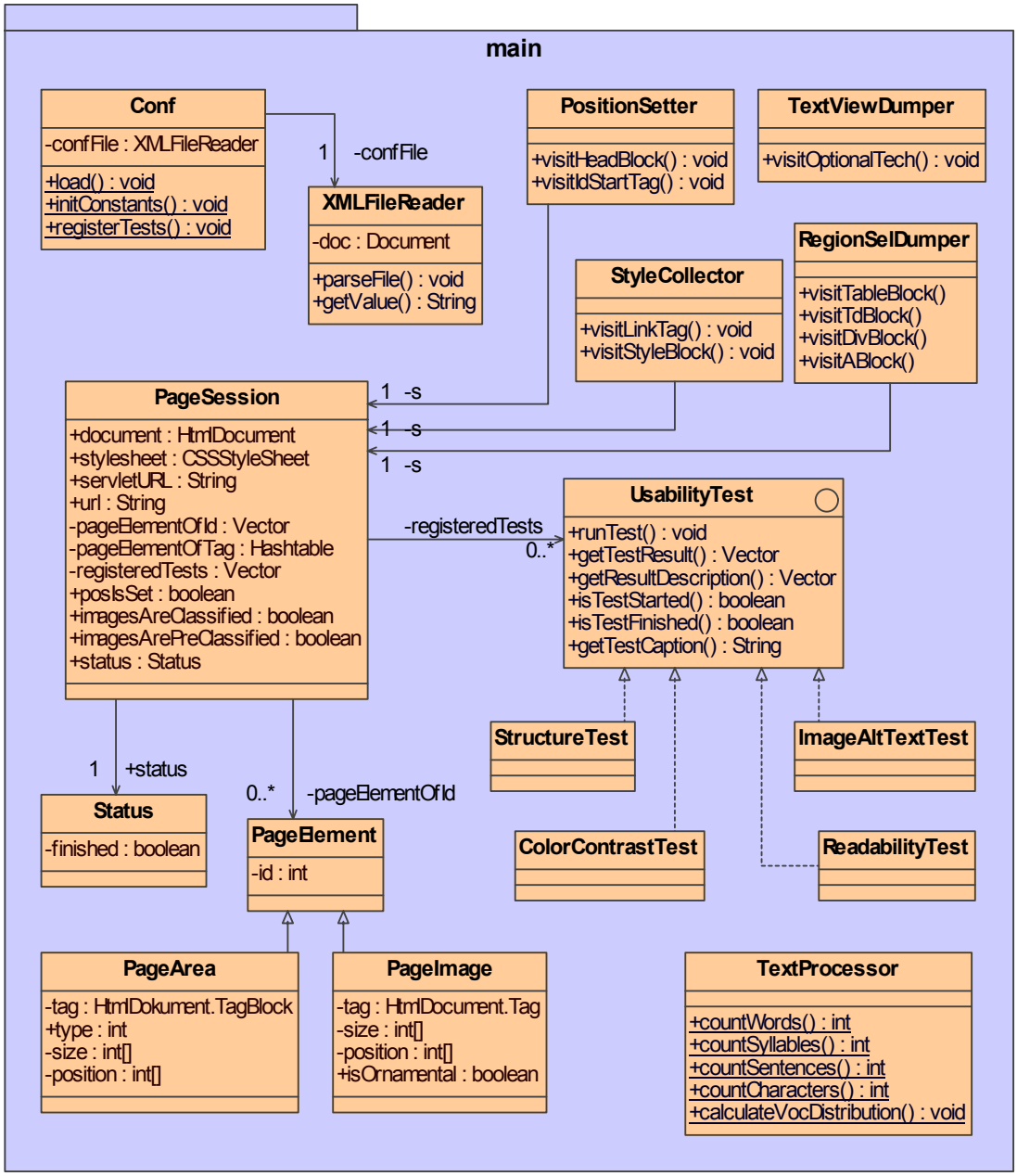


Abbildung 5.7: Klassendiagramm des erweiterten Prototyps

Ebenfalls neu sind eine Reihe von Unterklassen des HtmlVisitors aus dem Parser-Paket: PositionSetter, StyleCollector und TextViewDumper. Diese werden benötigt, um zusätzliche Manipu-

lationen auf dem Dokument-Modell der Seite durchzuführen - beispielsweise um an die Positionsdaten von Seitenelementen zu kommen oder eine reine Textansicht des Dokuments zu generieren.

Neben einigen Hilfsklassen zur Ablaufsteuerung oder für bestimmte Analysen, gibt es dann noch zwei Klassen, die dafür sorgen, dass der neue Prototyp zentral über eine XML-Datei konfiguriert werden kann (Conf, XMLFileReader). Sämtliche Konstanten und Zusatzdateien, die der Evaluator benötigt, werden in Zukunft in dieser XML-Datei spezifiziert. Außerdem soll es dort möglich sein, bestimmte Testfälle zu aktivieren oder zu deaktivieren.

5.3.2 Funktionsweise

Nachfolgend wird der grundsätzliche Ablauf einer Evaluierung erläutert. Die groben Zusammenhänge sind in dem Sequenzdiagramm in Abbildung 5.8 unten ersichtlich.

Nach dem Aufrufen der Anwendung im Browser, wird der Benutzer zunächst zum Eingeben der URL, die überprüft werden soll, aufgefordert (index.jsp). Nachdem diese URL an den Tomcat-Server übermittelt wurde, erstellt dieser ein PageSession-Objekt zu dem Aufruf und lädt den HTML-Quellcode der Zielseite herunter (Sequenzdiagramm, 1-4). Dieser Quellcode wird in Form eines Streams an den HtmlParser übergeben, der daraus ein Document Object Model generiert, welches in der PageSession abgespeichert wird. Zusätzlich werden in der Seite vorhandene CSS-Informationen gesammelt und von dem, neu in das Projekt integrierten, CSS-Parser verarbeitet und gespeichert (Sequenzdiagramm, 5-6). Das Dokument wird gleich darauf von dem PositionSetter durchlaufen, der für alle relevanten Elemente auf der Seite ein entsprechendes PageElement-Objekt erstellt und die ID dieses Objekts in Form eines class-Eintrages in den entsprechenden Tag des HTML-Codes schreibt. In diesen veränderten Code wird außerdem noch ein Javascript zur Positionsermittlung eingefügt. Dann erst wird der veränderte Quellcode der Seite an den Browser zurückgeschickt (Sequenzdiagramm, 7-8). Zusätzlich ist anzumerken, dass nach dem Absenden der URL im Browser ein Frameset geöffnet wird und der mit dem Positions-Script versehene Quellcode in einem versteckten, minimierten Frame aufgerufen wird (positionSetter.jsp). Die Ausgabe des Haupt-Frames wird während dieser Zeit serverseitig von einem Status-Objekt blockiert (pleaseWait.jsp).

In dem minimierten Frame wird inzwischen das Javascript, das in die HTML-Response eingefügt wurde, aktiv (onload) und stellt die Größen- und Positionsdaten von allen Elementen fest, die im Quellcode zuvor mit einer ID versehen wurden. Das Javascript-Programm verknüpft diese Daten mit den entsprechenden IDs und sendet diese über einen asynchronen Aufruf von addPosition.jsp als POST-Daten zurück an den Server. Dort werden die Daten geparst und mit Hilfe der IDs den passenden PageElement-Objekten zugeordnet (Sequenzdiagramm, 9-11). Nach vollständiger Verarbeitung der Positions- und Größenangaben wird außerdem die Blockierung durch das Status-Objekt aufgehoben, wodurch die Ausgabe des Haupt-Frames beginnen kann (regions.jsp). Mit Hilfe des RegionSelDumper wird die Ausgabe so modifiziert, dass der Benutzer über kleine Icons Bereiche auswählen und typisieren kann. Außerdem werden Links in das Dokument eingefügt, mit deren Hilfe man auf der Seite vorhandene Grafiken klassifizieren kann (Sequenzdiagramm, 12-13). Die so annotierte Seite ist die erste, für den Benutzer gedachte, sichtbare Antwort des Servers nach dem Abschicken der URL am Anfang. Die Positionsermittlung, die über Javascript in dem minimierten Frame im Hintergrund stattgefunden hat, ist eine Zwischenlösung. Für eine fortgeschrittene Version des Evaluators wäre es sinnvoll, wenn diese Daten durch einen internen Render-Vorgang berechnet werden könnten. Im Zuge dieser Arbeit wird das allerdings nicht möglich sein, da dafür quasi ein eigenständiger, CSS-fähiger Browser implementiert und eingebunden werden müsste.

Im 2. Schritt der Evaluierung kann der Benutzer nun Zusatzinformationen in die Seite einfügen. Wie bereits erwähnt, sind solche Informationen zum Beispiel Typenangaben zu Seitenbereichen

oder die Klassifikation von Bildern. Im Sequenzdiagramm in Abbildung 5.8 unten entspricht dies den Punkten 14 bis 16.

Wenn dieser Zwischenschritt abgeschlossen ist, startet der Benutzer die eigentliche Analyse der Seite, wobei nun auf die zusätzlichen Metainformationen zurückgegriffen werden kann. Abhängig von den Einstellungen bindet result.jsp dynamisch alle aktivierten Testfälle ein, startet diese und fügt das Ergebnis, das sie zurückliefern, in die HTML-Response ein (Sequenzdiagramm, 17-21).

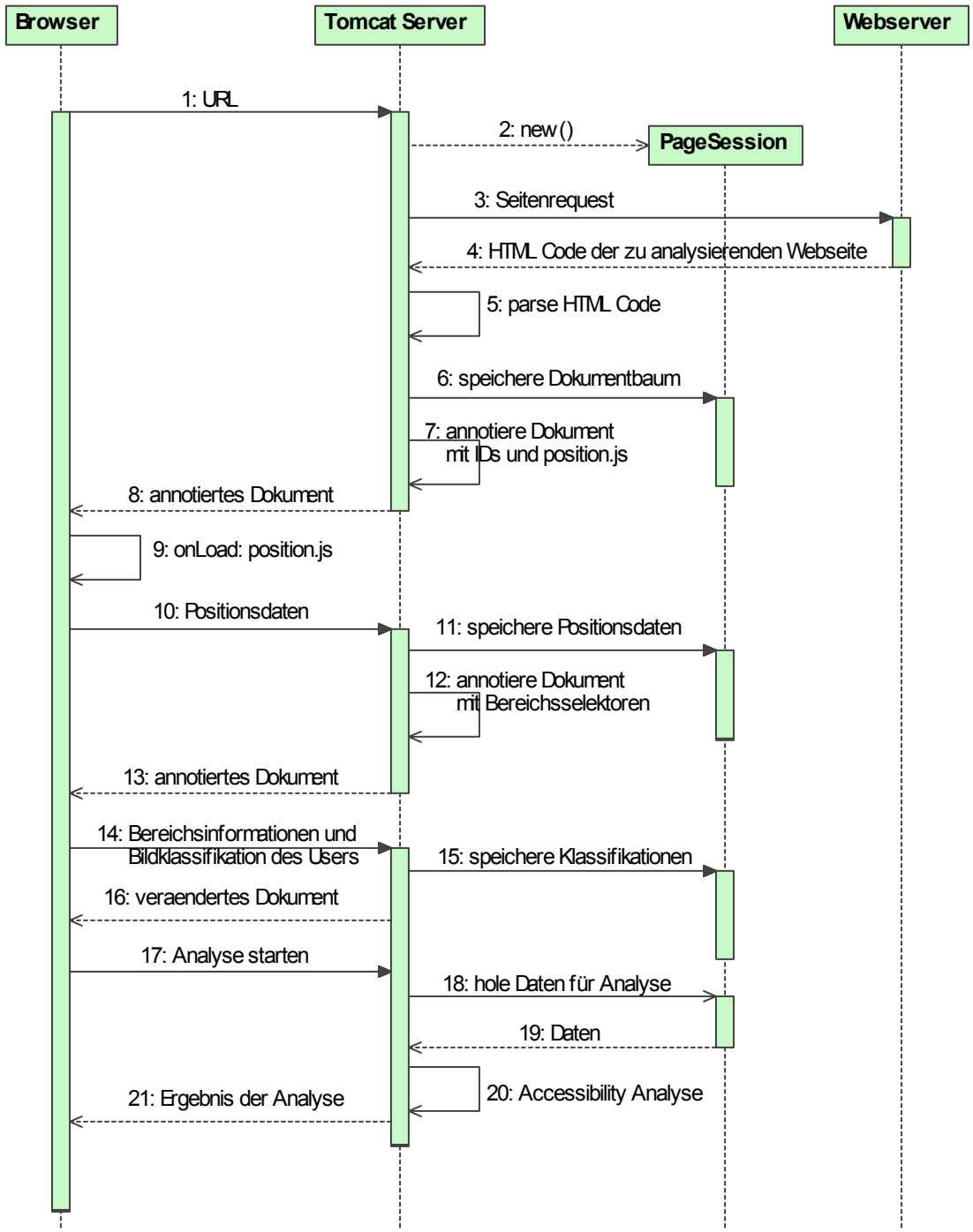


Abbildung 5.8: Sequenzdiagramm - Ablauf einer Seitenanalyse

6 Implementierung

Dieser Abschnitt beschäftigt sich mit den Implementierungsdetails der Hauptbereiche des in Kapitel 5 eingeführten, erweiterten Prototypen. Zuerst wird die Vorgehensweise bei der Analyse der Alternativtexte erläutert. Danach werden Details der Readability- und Farbkontrastanalyse betrachtet und schließlich wird noch kurz auf die Analyse der Seitenstruktur eingegangen.

6.1 Analyse der Alternativtexte von Bildinhalten

Die Überprüfung der Alternativtexte gliedert sich im neuen Prototyp in zwei Schritte: Zu Beginn werden die vorhandenen Bilder in Klassen eingeteilt, die festlegen, ob das Bild einen relevanten Inhalt hat oder nur dekorative Zwecke erfüllt. Erst danach werden die vergebenen Alt-Texte analysiert. Die Bildklasse beeinflusst dabei das weitere Verfahren der Analyse.

6.1.1 Bildklassifizierung

Die benötigten Metainformationen zu den Bildern werden auf folgende Arten gewonnen: Zum Einen versucht der Prototyp aus der Position und Größe der Grafiken im Quelltext Rückschlüsse auf deren Zweck zu ziehen. Dazu werden die entsprechenden Daten der PageImage-Objekte verwendet, die beim Parsen der Seite erstellt wurden. Während des ersten Ladevorgangs der annotierten Seite durch den RegionSelDumper, wird dann überprüft, ob das Bild eine bestimmte Größe nicht überschreitet (konfigurierbar) und auch nicht innerhalb eines Links vorkommt. Solche Bilder werden häufig zu Layoutzwecken „missbraucht“, um zum Beispiel eine Tabellenzelle auf eine bestimmte Größe auszudehnen. Diese Art von Bildern wird „spacer image“ genannt und kann als dekorativ eingestuft werden. Sehr oft sind diese Grafiken auch transparent. Der einfache Test könnte deshalb in Zukunft darauf erweitert werden, dass verdächtige Grafiken vom Evaluator zusätzlich auf vollständige Transparenz überprüft werden (image.getColorModel().getAlphas() bzw. image.getColorModel().getTransparentPixel()).

Auf diese Art wird eine Vorklassifizierung erzeugt, die nun im Zwischenschritt der Evaluation dem Benutzer präsentiert wird. Als dekorativ eingestufte Bilder werden dabei rot schattiert dargestellt, damit der Benutzer auf einen Blick sehen kann, welche Bilder betroffen sind und gegebenenfalls Ergänzungen oder Änderungen vornehmen kann. Das entspricht auch der zweiten Möglichkeit, wie die Bilder eingeteilt werden können: Die Grafiken werden bei der Ausgabe der Seite mit Links hinterlegt und wenn der Benutzer nun auf ein Bild klickt, wird die Seite mit der ID der Grafik als Parameter neu geladen und der Server ändert dabei die entsprechende Klassifikation und gibt die neue Ansicht aus. Das wird wiederum durch den RegionSelDumper abgewickelt. Damit dieser nicht bei jedem Aufruf versucht, die Bilder erneut automatisch zu klassifizieren und dadurch Änderungen des Benutzers überschrieben werden, wird nach dem ersten Laden der Seite in der PageSession das Flag imagesAreClassified = true gesetzt. Abbildung 6.1 unten zeigt ein vom Benutzer als dekorativ markiertes Bild (rechts), sowie den Tooltip-Text, der erscheint, wenn die Maus auf das nicht-dekorative Bild links davon geführt wird. Die rote Schattierung der Bilder wird mit einem halbtransparenten Frame realisiert, der

über die Position der ursprünglichen Grafik gelegt wird. Auch zu erkennen sind kleine rote Schattierungen, die vom Programm selbst gesetzt wurden und „spacer images“ anzeigen.

Abbildung 6.1: Klassifizierung von Bildern durch Mausklick

Schließlich besteht noch die Möglichkeit, Metainformationen zu verwenden, die bereits bei der Erstellung der Webseite von entsprechenden Autorenwerkzeugen abgefragt und in den Quelltext integriert wurden. Dies kann zwar nicht vorausgesetzt werden, stellt aber durchaus eine realistische Möglichkeit zur zusätzlichen Informationsgewinnung dar und wird deshalb prototypisch gezeigt. Beim ersten Parsevorgang der Seite prüft der Evaluator, ob solche Informationen im Code vorhanden sind. In unserem Fall wird dabei nach bestimmten Werten im class-Attribut der Elemente gesucht. Dort könnte die Information, dass es sich um eine dekorative Grafik handelt, wie folgt eingebaut werden: . Wenn das Prüfprogramm auf eine solche Stelle trifft, dann wird in der PageSession das Flag imagesArePreClassified = true gesetzt. Dies verhindert, dass das Programm versucht, die Vorklassifizierung der Bilder später durch eigene Kriterien zu überschreiben. Entsprechend präparierte Webseiten können auf diese Art sehr genau und schnell geprüft werden. Wenn diese Zusatzinformation bereits von Webeditoren beim Einfügen von Bildern abgefragt und eingefügt wird, entsteht dadurch nicht einmal ein merklicher Mehraufwand. In jedem Fall sollte dies aber weit effektiver sein, als eine nachträgliche Klassifizierung (durch eine andere Person) während der Evaluierung. Das muss aber erst durch weitere, konkrete Untersuchungen bestätigt werden.

6.1.2 Alternativtext-Test

Der in der Analysephase folgende Alternativtext-Test holt sich zuerst alle PageElement-Objekte aus der Session und durchläuft diese. Immer wenn es sich um ein Objekt vom Typ PageImage handelt, wird dieses gecastet und der zugehörige Tag aus dem Dokument-Baum abgerufen. Die Attributliste des Tag-Objekts wird danach auf Vorkommen von „src“, „alt“ und „longdesc“ geprüft. Wenn kein „alt“-Attribut vorhanden ist, kann sofort ein Fehler ausgegeben werden. Andernfalls wird nun geprüft zu welcher Klasse das aktuelle Bild gehört. Handelt es sich um ein dekoratives Bild (pi.isOrnamental = true), dann führt ein nicht leerer ALT-Text zu einem Fehler. Bilder mit explizit leerem ALT-Text werden an dieser Stelle als korrekt gezählt. Danach überprüft der Algorithmus, ob es sich um ein inhaltlich relevantes Bild mit leerem ALT-Text handelt und gibt ebenfalls einen Fehler aus, wenn dies so ist. Alle übrigen Bilder fallen in die Kategorie „relevant und irgendein ALT-Text vorhanden“. Diese Kategorie ist nach wie vor kaum verlässlich zu beurteilen. Es können zwar weiterführende Tests auf dem Alternativtext durchgeführt werden, als Resultat werden aber meistens nur noch Warnungen ausgegeben. So zum Beispiel, wenn der Alternativtext einer Grafik deren Dateinamen enthält, was auf einen automatisch gesetzten, schlechten Text durch einen Webeditor hindeutet. Auch die Länge des Alternativtexts kann hier überprüft werden, denn sehr lange Beschreibungen sollten im „longdesc“-Attribut referenziert werden.

6.2 Readability-Analyse

Bei der Untersuchung der Lesbarkeit von Textpassagen der analysierten Webseite werden mehrere Verfahren miteinander kombiniert: Zum Einen werden in Kapitel 4.1 vorgestellte Readability-Kennzahlen zur Analyse der Textstruktur implementiert, die mit den inhaltlichen Aussagen von Wortschatzanalysen kombiniert werden. Gekapselt werden diese Funktionalitäten in der Klasse ReadabilityTest, die von HtmlVisitor erbt und zusätzlich das Interface UsabilityTest implementiert, das für eine einheitliche Schnittstelle für alle Testfälle sorgt.

6.2.1 Identifikation von relevanten Textpassagen

Das Problem vorhandener Readability-Formeln besteht darin, dass die meisten für klassische Texte in Satzform entwickelt worden sind und es momentan noch keine Verfahren gibt, die speziell auf die Webstruktur von Texten abgestimmt sind. Webtexte werden so verfasst, dass sie schnell erfasst („gescannt“) werden können, sie sind im Allgemeinen kürzer und verwenden viele Strukturierungsobjekte wie Listen, Überschriften und Hervorhebungen. Scannt man nun nur stur nach Satztrennzeichen und liest ungeachtet dieser Umstände sonst alles ein, dann verfälscht sich die Satzlänge und damit auch das Ergebnis von Kennzahlen, die darauf beruhen. Um diesen Nachteil auszugleichen, wird der Quellcode der Webseite zuerst mit einem Algorithmus geparst, der Textpassagen extrahiert, die sich in Satzform befinden.

Dazu wird das Dokument-Modell der Eingabewebseite mit einem HtmlVisitor durchlaufen, der Schritt für Schritt alle Elemente besucht und dabei die relevanten Textstellen herausfiltert. Abbildung 6.2 unten zeigt das Aktivitätsdiagramm dieses Prozesses.

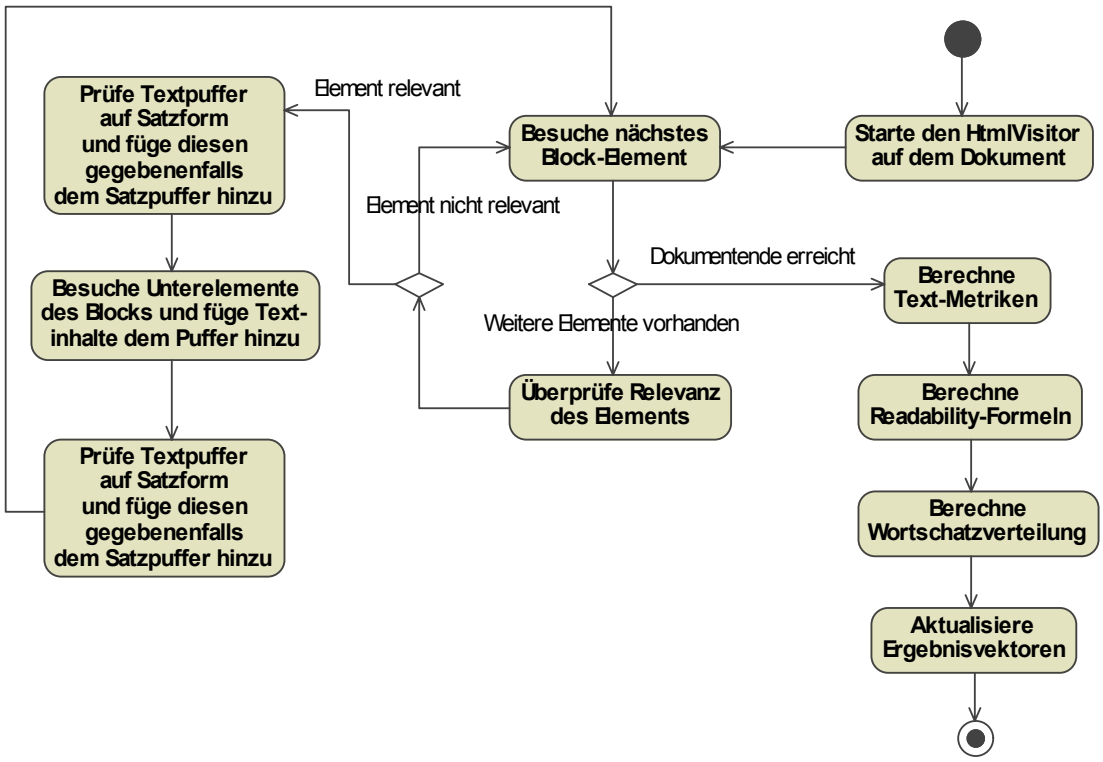


Abbildung 6.2: Aktivitätsdiagramm ReadabilityTest

Beim Besuchen eines Block-Elements wird dabei zunächst entschieden, ob man sich bereits innerhalb des <body> Abschnitts der Webseite befindet. Alle Abschnitte davor oder danach stellen keine Bereiche dar, die beim Rendern der Webseite in der Ausgabe erscheinen und können deshalb pauschal ignoriert werden. Ebenfalls komplett (inklusive aller Unterelemente) aus-

genommen werden die Tag-Blöcke <applet>, <object>, <script> und <style>, da diese externe Anwendungen/Objekte einbinden oder nur Code enthalten, der nicht ausgegeben wird und zudem keine natürliche Sprache darstellt, die geprüft werden soll. Ähnlich verhält es sich mit dem Inhalt von <code> Blöcken. Diese definieren Codebeispiele, die zwar in der Ausgabe der Webseite enthalten sind, sie sind aber nicht für die Readability-Prüfung des natürlichsprachlichen Textes der Seite relevant. Auch ignoriert werden <select> Blöcke, die theoretisch beliebig lange und beliebig viele Text-Einträge für ein Kombinationsfeld enthalten können und einen Satz sehr stark verlängern und verfälschen können.

Alle anderen Blöcke werden normal durchlaufen und gegebenenfalls darin enthaltene Textabschnitte werden in einem Textpuffer gesammelt. Beim Durchlaufen jedes Tag-Blocks wird außerdem geprüft, ob es sich um ein echtes Html-Absatzelement handelt. Ein Absatzelement markiert einen in sich abgeschlossenen, logischen Bereich und veranlasst den Browser dazu, einen neuen Absatz zu rendern. Das heißt, sowohl beim Betreten als auch beim Verlassen eines solchen Elements, kann jeweils geprüft werden, ob sich der inzwischen im Puffer angesammelte Text in Satzform befindet. Wenn dem so ist, dann wird der Inhalt des Puffers in einen Speicherbereich übertragen, der den gesamten relevanten Text der Seite sammelt. Danach wird der Puffer geleert und die nachfolgenden/inneren Blöcke werden betrachtet.

Würde man im Gegensatz dazu beim Betreten oder Verlassen eines Absatzelements den Pufferinhalt nicht direkt auf Satzform prüfen, dann würden z. B. Überschriften, Listen oder Inhalte anderer Tabellenzellen zum Puffer hinzugefügt, die später fälschlicherweise einen Satz verlängern, zu dem sie gar nicht gehören. Absatzelemente sind unter anderem <p>, <h1> bis <h6>, <pre> oder <table>.

Handelt es sich bei einem Block um kein Html-Absatzelement, sondern um ein sogenanntes Inline-Element, dann kann der Text im Puffer noch nicht auf Satzform geprüft werden, da man sich möglicherweise noch mitten in einem unvollständigen Satz befindet. Beispiele für Inline-Elemente wären , <font> oder <a> Blöcke, mit denen man bestimmte Abschnitte im Textfluss formatieren oder als Link deklarieren kann.

Am Ende hat der Algorithmus also alle Textteile, die in ganzen Sätzen formuliert sind, gesammelt und für weitere Analysen zwischengespeichert. Gleichzeitig wurde für die satzunabhängige FORCAST-Formel oder für spätere Wortschatztests eine Version des Textes gespeichert, der natürlichsprachliche Wörter aus allen relevanten Bereichen enthält, ohne auf die Satzform Rücksicht zu nehmen. Dazu zählt jetzt beispielsweise auch der Textinhalt von <select> Bereichen.

6.2.2 Silbenzählung

Zur Bestimmung der Silbenzahl in einem Wort wird ein sehr vereinfachter Algorithmus verwendet, der auf der Anzahl der vorkommenden Vokale basiert. Dabei wird das Wort Zeichen für Zeichen durchlaufen und getestet, ob es sich bei dem aktuellen Buchstaben um einen Vokal handelt. Trifft dies zu wird eine Silbe gezählt und ein Flag gesetzt, dass der letzte Buchstabe ein Vokal war. Das Flag ist deswegen nötig, damit nicht mehrere Vokale in Folge als eigenständige Silbe behandelt werden, sondern die Silbenanzahl erst dann wieder inkrementiert wird, wenn zwischen zwei Vokalen ein Konsonant gefunden wurde.

Wenn alle Buchstaben des Wortes abgearbeitet wurden, wird noch geprüft, ob es sich beim letzten Zeichen um ein „e“ gehandelt hat. Falls dem so ist, wird die endgültige Silbenanzahl noch um eins dekrementiert, da es sich im Englischen mit hoher Wahrscheinlichkeit um ein „stummes e“ am Wortende handelt und stumme Vokale im Allgemeinen keine neue Silbe bewirken.

Wie bereits erwähnt, handelt es sich bei dem Silbenzähler um einen sehr primitiven Algorithmus. Diese Funktion muss für exaktere Ergebnisse unbedingt weiter verfeinert werden. Denkbar

wäre auch eine datenbankgestützte Lösung, die genaue Silbenzahlen für eine umfassende Wortsammlung (vor allem Ausnahmefälle) enthält. Solche Daten lagen aber für diese Arbeit nicht vor und hätten außerdem den Rahmen gesprengt, da ja nur eine prototypische Verbesserung der Ausgabe von Accessibility-Prüfprogrammen mittels Readability-Informationen gezeigt werden sollte.

6.2.3 Wort- und Satzzählung

Um die Anzahl der Wörter pro Satz oder die Anzahl an Sätzen in einem Text-String festzustellen, wurde die Java BreakIterator-Klasse verwendet. Diese Klasse enthält eine Sammlung von Methoden, die dafür gedacht sind verschiedenste Grenzen und Umbrüche in Texten zu finden. Über den Aufruf von Factory-Methoden erhält man spezielle Instanzen der Klasse, die zum Beispiel für Wort-, Zeilen- oder Satzumbrüche zuständig sind. Der Factory-Methode kann dabei ein Locale-Objekt übergeben werden, das dafür sorgt, dass Trennzeichen und –regeln aus der gewünschten Region verwendet werden.

Auflistung 6.1 unten zeigt die Methode countWords(string), die einen BreakIterator verwendet, um die Wörter in dem übergebenen String zu zählen.

```
1  public static int countWords(String s){
2     int count = 0;
3     BreakIterator iterator = BreakItera-
   tor.getWordInstance(Locale.ENGLISH);
4     iterator.setText(s);
5     int current = iterator.current();
6     while (current < s.length()){
7        if (Character.isLetter(s.charAt(current))) {
8           count++;
9           int begin = current;
10          current = iterator.next();
11       }
12       else current = iterator.next();
13    }
14    return count;
15 }
```

Auflistung 6.1: Definition der Methode TextProcessor.countWords()

Zuerst wird eine englische Wort-Instanz des BreakIterators generiert an diese der Eingabetext übergeben wird. setText(s) bewirkt außerdem, dass direkt die Position des ersten Wortanfangs gesucht wird, die man anschließend über iterator.current() abfragen kann. Danach wird der Eingabestring solange durchlaufen, bis innerhalb seiner Zeichenlänge kein neuer Wortanfang mehr gefunden werden kann. Als Wörter werden zudem nur Teilstücke des Strings gezählt, die mit einem Buchstaben beginnen. Character.isLetter(char) bietet eine vordefinierte Möglichkeit, dies zu überprüfen. Am Ende einer Iteration wird mit current = iterator.next() der Iterator auf die Position des nächsten Wortanfangs gesetzt und die Schleife beginnt von vorn.

Auf die selbe Weise können auch einzelne Zeichen, Sätze oder Wörter mit einer bestimmten Anzahl an Silben (in Kombination mit der Silbenzählfunktion) gezählt werden. Sämtliche Methoden zum Berechnen dieser Textmetriken sind static und wurden in der Klasse TextProcessor gesammelt.

6.2.4 Wortschatzanalyse

Wenn der ReadabilityTest in der Konfiguration des Evaluators aktiviert ist, dann werden gleich direkt im Konstruktor auch alle Wortschatzdateien eingelesen, die in der XML-

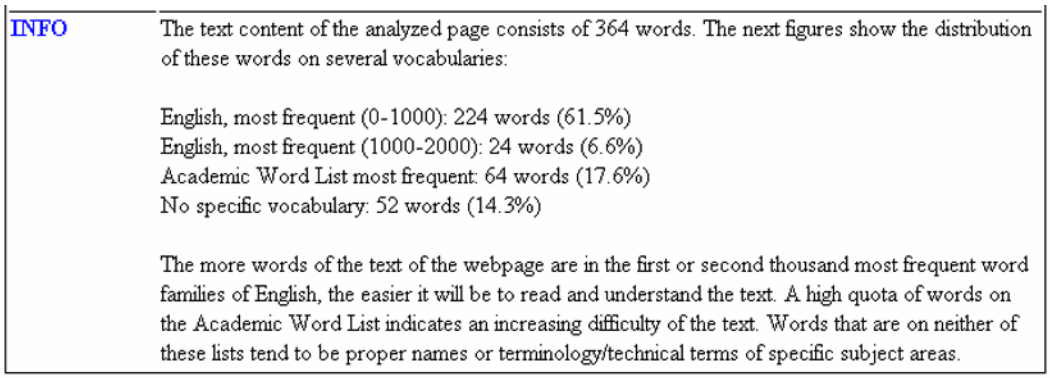
Konfigurationsdatei angegeben wurden. Die Methode parseFileToHashtable(filename) öffnet dafür einen gepufferten InputStreamReader, der die Dateien zeilenweise einliest. In jeder Zeile wird mit einem regulären Ausdruck festgestellt, ob ein Wort vorhanden ist und wenn dies zutrifft, wird das Wort ausgeschnitten und als neuer Schlüssel in eine Hashtabelle eingefügt, die für diese Wortschatzdatei erstellt wird. Die auf diese Weise erstellten Hashtabellen für jeden einzelnen Wortschatz werden dann in einem Vektor abgelegt und für weitere Analysen bereitgehalten. Standardmäßig verfügt der Evaluator über drei Wortschatzlisten:

1. Die 1000 häufigsten Wortfamilien, die im Englischen verwendet werden.
2. Die 1000-2000 häufigsten Wortfamilien, die im Englischen verwendet werden.
3. Die 570 häufigsten Wortfamilien in englischen, akademischen Texten, die aber nicht bereits durch die 2000 häufigsten englischen Wortfamilien abgedeckt sind (The Academic Word List, vergleiche Kapitel …).

Insgesamt besteht jede dieser Dateien aus ungefähr 4000 bis 5000 einzelnen Einträgen.

Nach dem Starten des Readability-Tests in der Auswertungsphase des Evaluators, wird jedes einzelne Wort der analysierten Webseite mit den vorhandenen Wortschatzlisten verglichen und diesen zugeordnet. Wörter, die in keiner dieser Wortlisten vorkommen, werden extra gezählt.

Der Test berechnet die prozentmäßige Verteilung des überprüften Textes auf die verschiedenen Wortschätze beziehungsweise den Anteil der Wörter, die nirgends enthalten sind. Je mehr Vokabular eines Texts sich mit den häufigsten Wortfamilien der entsprechenden Sprache deckt, desto einfacher ist der Text. Je größer die Anteile des für Akademiker typischen Vokabulars werden, desto mehr spezialisiert sich der Text auf eine Zielgruppe mit höherer Bildung. Der Anteil des Texts, der auf keiner der Listen vorkommt, muss nicht zwingend sehr seltene oder schwierige Wörter enthalten. Es kann sich einfach um Eigennamen oder spezielles (aber durchaus geläufiges) Vokabular eines Fachgebietes handeln, zu dem die analysierte Webseite gehört (vergleiche Abbildung 6.3 unten).



INFO	The text content of the analyzed page consists of 364 words. The next figures show the distribution of these words on several vocabularies: English, most frequent (0-1000): 224 words (61.5%) English, most frequent (1000-2000): 24 words (6.6%) Academic Word List most frequent: 64 words (17.6%) No specific vocabulary: 52 words (14.3%) The more words of the text of the webpage are in the first or second thousand most frequent word families of English, the easier it will be to read and understand the text. A high quota of words on the Academic Word List indicates an increasing difficulty of the text. Words that are on neither of these lists tend to be proper names or terminology/technical terms of specific subject areas.

Abbildung 6.3: Ergebnis einer Wortschatzanalyse

6.2.5 Kennzahlberechnung

Die eigentliche Berechnung der Kennzahlen stellt dann kein großes Problem mehr da. Es werden einfach die benötigten Parameter für die in Kapitel 4.1 vorgestellten Formeln ermittelt und das Gesamtergebnis entsprechend berechnet. Da alle Kennzahlen einen Wert im Schulstufensystem (Grade Level) ausgeben, wird davon noch der Durchschnitt berechnet, um dem Benutzer des Tools auf einen Blick zu zeigen, wie die Formeln „insgesamt“ entschieden haben. Der resultierende Schulstufenwert wird zur Veranschaulichung auch noch in einen ungefähren Altersbereich und die Stufe des Bildungssystems umgewandelt. Abbildung 6.4 unten zeigt die Ausgabe des Prototyps nach einer Readability-Analyse.

Results of Readability analysis

RESULT	DESCRIPTION
INFO	The text content has been analyzed with the following readabilty-formulas: Flesch-Kincaid Grade Level: 13.15 Gunning FOG Grade Level: 16.4 SMOG Grade Level: 14.74 Coleman-Liau Grade Level: 16.46 Automated Readability Index Grade Level: 13.36 FORCAST Grade Level: 12.51 The average Grade Level would be 14.44. This means that a visitor of the webpage needs approximately 14.44 years of education to be able to read and understand the text on the page. Therefore the target group should at least be at the age of 18 to 22, having a College/University level education.

Abbildung 6.4: Ergebnis einer Readability-Analyse

6.3 Farbkontrastanalyse

Für die Analyse der Farben und deren Kontrastverhältnis war es nötig, einen CSS-Parser in den Prototyp zu integrieren Das Verfahren unterteilt sich dann in einen Durchlauf zum Sammeln der CSS-Informationen (StyleCollector) und in die eigentliche Analyse der Farbgebung der Seite.

6.3.1 StyleCollector

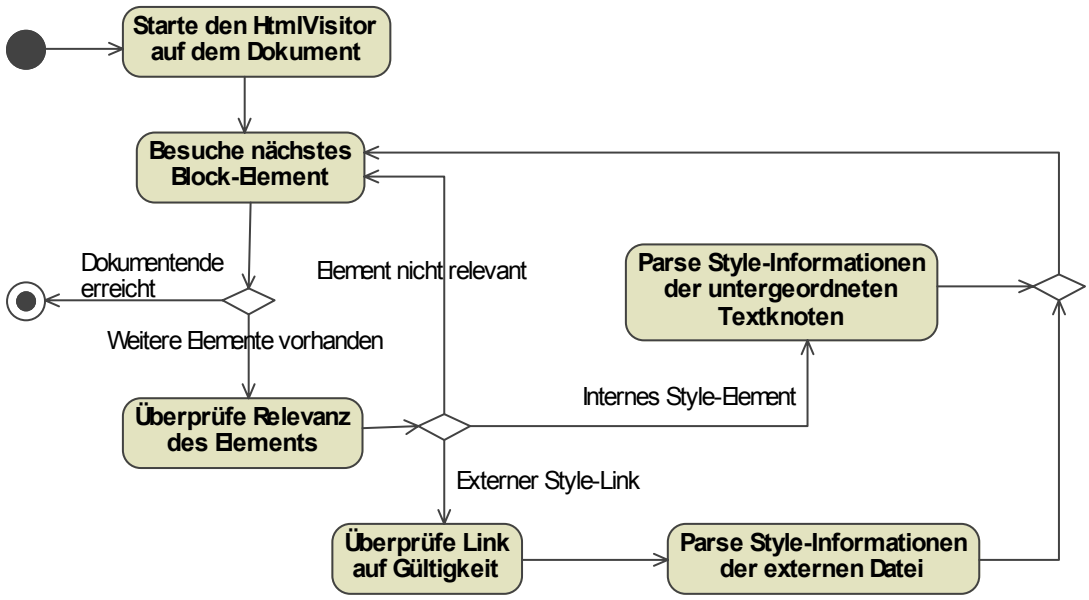


Abbildung 6.5: Aktivitätsdiagramm StyleCollector

Da es verschiedenste Möglichkeiten gibt, Style-Informationen in ein Html-Dokument einzubauen, wurde zuerst ein HtmlVisitor geschrieben, der dafür zuständig ist, alle möglichen globalen, dokumentweit gültigen Style-Informationen zu sammeln und in einem gemeinsamen CSSStylesheet-Objekt in der PageSession zu speichern. Dieser Visitor, der StyleCollector, durchläuft also das Dokument und sucht nach <link> oder <style> Tags, die CSS-Code enthalten können. Eigentlich dürfen beide Tags laut Spezifikation nur in dem <head> Block einer Html-Seite vor-

kommen, jedoch hat sich schnell gezeigt, dass ein solch restriktiver, spezifikationskonformer Parse-Vorgang für praktische Zwecke nicht ausreicht. Sehr häufig verwenden Webautoren <style> Blöcke im gesamten Dokument und dies wird von den Browsern auch akzeptiert. Um brauchbare Ergebnisse (das heißt, die selben Ergebnisse die ein Browser liefert) bei der Auswertung der Farben zu bekommen, war es also notwendig, sich dieser Ungenauigkeit anzupassen.

Abbildung 6.5 oben zeigt in einem Aktivitätsdiagramm die grundsätzliche Vorgehensweise des StyleCollectors. Dem ist hinzuzufügen, dass CSS-Informationen aus <style> Blöcken innerhalb des Html-Dokuments (interne Style-Elemente) vorerst in einem Puffer gesammelt werden, da diese erst nach Verlassen des <head> Blocks geparst und dem CSSStylesheet-Objekt hinzugefügt werden. Das hat den einfachen Grund, dass sich Stylesheet-Definitionen überlagern können und die Gültigkeit hängt dann von folgenden Kriterien ab:

- Globale Style-Informationen können über <style> Blöcke oder über externe Dateien (<link rel="stylesheet" … />) eingebunden werden. Sie gelten für das gesamte Dokument. Definitionen innerhalb von <style> Blöcken im HTML-Dokument haben dabei im Konfliktfall Vorrang gegenüber Definitionen in extern referenzierten Dateien.
- Jedes Element kann über ein style-Attribut lokale, für das Element gültige Style-Informationen enthalten. Diese wiederum überlagern im Konfliktfall die globalen CSS-Definitionen.
- Außerdem gilt für alle Arten von Deklarationen zusätzlich eine chronologische Überlagerung. Das bedeutet, wenn man für das selbe Element zum Beispiel zwei verschiedene Werte für die selbe Eigenschaft definiert, dann gilt der zuletzt definierte Wert.
- Html-spezifischen Formatierungselementen oder –attributen (<font>, color="…", bgcolor="…") kommt die geringste Bedeutung zu. Sie werden im Konfliktfall von allen erwähnten Style-Definitionen überschattet.

Für das im Prototyp verwendete CSSStylesheet-Objekt, das sämtliche globalen Style-Informationen enthält, ist es deswegen wichtig, dass alle CSS-Definitionen entsprechend der Überlagerungsregeln in der richtigen, chronologischen Reihenfolge eingefügt werden, so dass immer die letzte Definition Gültigkeit hat. Deshalb werden allfällige, externe CSS-Dateien zuerst eingebunden und erst nach dem Verlassen des <head> Blocks werden gepufferte Formate aus <style> Blöcken geparst, die innerhalb des <head> Blocks eventuell schon vorher gefunden wurden.

```
1  StringReader sr = new StringReader(styles);
2  InputSource is = new InputSource(sr);
3
4  // if the PageSession has no stylesheet yet, create one
5  if (s.stylesheet == null) {
6    s.stylesheet = parser.parseStyleSheet(is);
7  }
8  // else append the actual style information to the existing one
9  else {
10   CSSStyleSheet tempstyle = parser.parseStyleSheet(is);
11   CSSRuleList templist = tempstyle.getCssRules();
12   for (int i = 0; i < templist.getLength(); i++) {
13   // insert all rules from the temporary list to the end of the
   pagesession-stylesheet
14     s.stylesheet.insertRule(templist.item(i).getCssText(),
   s.stylesheet.getCssRules().getLength());
15   }
16 }
```

Auflistung 6.2: Parsen von Style-Informationen

Auflistung 6.2 oben zeigt das Parsen und Hinzufügen von CSS-Formaten, die aus <style> Blöcken innerhalb des HTML-Dokuments ausgelesen wurden.

Der StyleCollector berücksichtigt momentan noch keine Style-Formate, die über @import Anweisungen direkt im CSS-Code referenziert werden. Außerdem werden nur Formate für den Bildschirm beachtet (media="screen"). Andere Medienangaben, wie zum Beispiel für Ausdrucke oder Projektionen, werden im Rahmen dieser Arbeit bei der Farbanalyse nicht betrachtet.

6.3.2 Analyse mittels Farben-Stack

Die eigentliche Analyse der Farbinformationen wird während der Auswertungsphase des Accessibility-Evaluators von der Klasse ColorContrastTest gesteuert. Diese implementiert wiederum die abstrakten Methoden eines HtmlVisitors, um damit das Dokument-Objektmodell der überprüften Webseite durchlaufen zu können. Da der im Prototyp ursprünglich verwendete Html-Parser seinen Elementen keine CSS-Farbinformationen zuordnen kann, wird das vom StyleCollector erstellte Stylesheet-Objekt jetzt dazu verwendet, die Farbeinstellungen der einzelnen Seitenbereiche festzustellen. Die Idee dabei ist, beim Durchlaufen des Dokuments einen Farben-Stack anzulegen, der beim Betreten eines neuen Blocks jeweils um eine Farbinformation erweitert wird. Diese hat dann im gesamten Block Gültigkeit und beim Verlassen des Blocks wird die Farbinformation wieder vom Stack entfernt. Damit wird die Vererbung der Style-Eigenschaften an untergeordnete Elemente modelliert. Kann für den aktuellen Tag-Block in den Style-Informationen keine neue Farbeinstellung gefunden werden, dann wird einfach das oberste Element des Stacks kopiert und erneut auf den Stack gelegt. Der aktuelle Block erbt sozusagen die Farbe seines Elternelements. Abbildung 6.6 unten veranschaulicht die Aktivitäten der Kontrastanalyse.

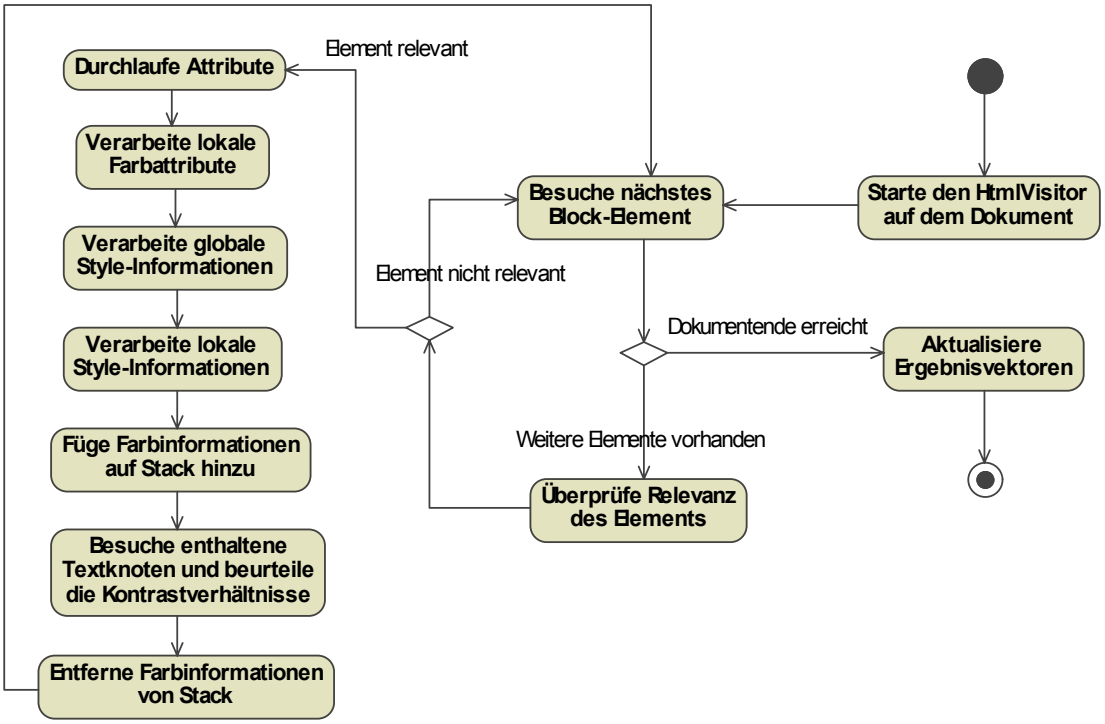


Abbildung 6.6: Aktivitätsdiagramm ColorContrastTest

Die Überprüfung der Relevanz eines Block-Elements bezieht sich dabei, ähnlich wie bei der Readability-Analyse, zum Beispiel auf <style> oder <script> Blöcke, die nicht näher betrachtet werden müssen, da deren Inhalt nicht am Bildschirm ausgegeben wird und somit auch irrelevant für die Analyse der Farbkontraste ist.

Von allen in Betracht kommenden Elementen werden anschließend die Attributlisten auf lokale Farb- oder Style-Attribute durchsucht. Einem temporären Farbobjekt werden dann zuerst eventuell vorhandene lokale, HTML-spezifische Farbinformationen zugeordnet. Dann werden globale und lokale Style-Informationen auf zutreffende Selektoren für das aktuelle Element durchsucht und das Farbobjekt wird gegebenenfalls aktualisiert. Die Reihenfolge dieser Verarbeitungsschritte drückt wieder die beim StyleCollector bereits erwähnte Gewichtung der CSS-Definitionen im Konfliktfall aus. Hat das temporäre Farbobjekt am Ende dieses Durchlaufs noch immer einen null-Wert, dann gab es für das vorliegende Element keine eigenen Farbdefinitionen und es erbt die Farbinformation seines Elternelements. Diese Informationen werden dann am Stack abgelegt. Auflistung 6.3 unten zeigt einen Ausschnitt des Codes, der diese Operationen auf dem Stack durchführt.

```
1  // now, add the final color information for this tagblock to the
   stack
2  // if nothing has changed, use the color of the parent element
3  if (!changedFGC) {
4    // if there's nothing on the stack yet, add the default color
5    if (foregroundColorStack.isEmpty()) {
6      foregroundColorStack.add(DEFAULT_FGC);
7    }
8    // else inherit the color of the parent element
9    else {
10     foregroundColorStack.add(foregroundColorStack.lastElement());
11   }
12 }
13 // else add the new color
14 else {
15   foregroundColorStack.add(fgc);
16 }
17
18 // visit elements and text within this block element
19 visit(bl.body);
20 visit(bl.endTag);
21
22 // at the end of each block, remove its color information again
23 foregroundColorStack.remove(foregroundColorStack.lastElement());
```

Auflistung 6.3: Manipulation des Farben-Stack

Eine der Hauptschwierigkeiten beim Feststellen der für das jeweilige Element zutreffenden Farbinformationen aus dem globalen Stylesheet, war die Verarbeitung der CSS-Selektoren. Dies musste manuell codiert werden, da das von dem CSS-Parser zurückgelieferte Stylesheet-Objekt eine solche Funktionalität leider nicht zur Verfügung stellt. Die CSS-Selektoren, die als Strings vorliegen, werden deswegen mit drei verschiedenen Methoden und regulären Ausdrücken auf passende Tagnamen, passende Klassennamen oder passende ID-Angaben durchsucht. Die letzte Definition (color: …, background-color: …, background: …), die in dieser Reihenfolge gefunden wird, hat das stärkste Gewicht und bestimmt die endgültige Farbe. Auflistung 6.4 unten zeigt einige Selektoren und Beispieldefinitionen in einem CSS, die sich gegenseitig überlagern.

```
1  /* sample html-codesnippet */
2  <h1 id="id1" class="header1">Testheader</h1>
3
4  /* sample css-definitions */
5  /* text color definition for the h1 tag */
6  h1 { color: #FF0000; }
7  /* this definition overrides the first one */
8  h1 { color: #00FF00; }
9  /* definitions for class-names are stronger */
10 h1.header1 { color: #0000FF; }
11 /* definitions for ids are stronger again */
12 h1#id1 { color: #AA0000; }
13 /* this definition would override the first two, but the class and
14    id definitions are stronger, so the color doesn't change */
15 h1 { color: #00AA00; }
```

Auflistung 6.4: CSS-Beispieldefinitionen und Überladung

Komplizierte Verschachtelungen von Element- und Klassennamen innerhalb der Selektoren oder Spezialselektoren wie + und > werden noch nicht unterstützt. Manche Browser unterstützen diese ebenfalls nicht vollständig.

Der eigentliche Kontrasttest, der für alle Textknoten in relevanten Tags durchgeführt werden muss, ist dann wieder schnell erledigt. Wenn sich der Visitor in einem Textelement befindet, holt er sich die jeweils letzten Farbelemente vom Vorder- und Hintergrund-Stack und prüft das Kontrastverhältnis der Farben gemäß den Vorgaben des W3C. Ist der errechnete Kontrast kleiner als der zulässige Minimalwert, so wird ein Ausschnitt des betroffenen Textes, inklusive der entsprechenden Farbformatierung zur Veranschaulichung, als Fehlermeldung ausgegeben (vergleiche Abbildung 6.7 unten).

Results of Color contrast analysis

RESULT	DESCRIPTION	
ERROR	The following text passages have a poor contrast ratio and may therefore be hard to read:	
	Contrast Ratio	Text snippet from the webpage
	4.0	If you have already registered for the 5
	4.0	If you have never registered for the 508
	4.0	Having problems with the 508 Universe?

Abbildung 6.7: Ergebnis einer Farbkontrastanalyse

Der Farbkontrast-Test hat momentan noch folgende Einschränkungen:

- Keine Überprüfung der Linkfarben (normal, aktiv, besucht) im Bezug auf den Hintergrund. Lösung: Anlegen weiterer spezifischer Farb-Stacks und entsprechende Überprüfung nur in <a> Elementen.
- Keine Überprüfung der Linkfarben (normal, aktiv, besucht) im Bezug auf die Farbe des angrenzenden Texts. Lösung: Jeweils die Farbinformationen des letzten Elements vor und nach einem Link im Puffer behalten und entsprechende Berechnungen damit durchführen. Damit könnte getestet werden, ob die Farbgebung des Links überhaupt ausreicht, um ihn optisch im normalen Textfluss hervorzuheben.

- Spezialfälle, wie zum Beispiel eine HTML-spezifische Angabe von Schriftart und –farbe mittels <basefont>, sind nicht implementiert. Solche Formatierungen werden zukünftig allerdings verschwinden, da solche Tags nicht mit der Trennung von Layout und Struktur konform sind.

Die bereits erwähnte Einschränkung bezüglich verschachtelter Selektoren sollte in Zukunft am besten über einen Parser gelöst werden, der sowohl den HTML- als auch den CSS-Code parsen kann und diese Informationen gleich miteinander verknüpft. Auch für andere Tests wäre es sicherlich sinnvoll, wenn der Evaluator in Zukunft die Webseite quasi intern selbstständig rendern kann und Größen- oder Positionsangaben von Elementen ohne externe Hilfsmittel (Javascript) und in beliebigen Auflösungen feststellen kann. Dies würde natürlich dem Entwickeln eines eigenständigen Browsers nahe kommen und kann im Rahmen dieser Arbeit nicht umgesetzt werden. Für zukünftige Arbeiten kann man sich dazu aber das Projekt des Java-XHTML/CSS2-Renderers „Flying Saucer“ (https://xhtmlrenderer.dev.java.net/) anschauen und Integrationspotentiale abklären.

6.4 Analyse der Seitenstruktur ohne CSS/Skripte

Eine automatisierte Analyse der Zugänglichkeit einer Seite, ohne aktivierte Zusatztechnologien wie CSS, Skriptsprachen, Farbinformationen oder Bilder, gestaltete sich erwartungsgemäß schwierig. Es wurden Möglichkeiten gesucht, um zum Beispiel die Positionierung einzelner Bereiche der Seite (Navigation, Hauptcontent, …) mit und ohne aktiviertem CSS-Layout nachzuverfolgen und Rückschlüsse auf eine brauchbare Strukturierung der Inhalte zu ziehen. Ein solches Tracking von logischen Einheiten der Seite wäre theoretisch durchaus möglich. Man könnte vielleicht auch Vermutungen anstellen, ob es sich negativ auswirkt, wenn sich beispielsweise die Navigation nach der Deaktivierung von CSS nicht mehr „gut sichtbar“ irgendwo im oberen Bereich der Webseite befindet, sondern plötzlich nach allen anderen Inhalten am Ende des Dokuments dargestellt wird. Leider ist es kaum möglich, hier generelle Aussagen zu treffen, weil es für einen Computer unmöglich ist, verlässlich zu sagen, ob eine Position in einem bestimmten Kontext sinnvoll ist oder eben nicht. Das Beispiel mit dem Navigationsbereich kann hier am ehesten noch mit schlüssigen Argumenten belegt werden. Vor allem für Benutzer von seriellen Ausgabegeräten (z. B. Screenreader) ist es untragbar, wenn die Möglichkeit auf der Seite weiterzunavigieren, erst nach der Ausgabe (dem Vorlesen) sämtlicher Inhalte möglich ist, da diese zuvor „dargestellt“ (angeordnet) werden.

Ein weiterer, einfach zu implementierender Test wäre die Suche nach Links, die Javascript-Befehle ausführen (<a href=“javascript: …“>), oder anderen Elementen, die Eventhandler (onClick, onLoad, …) mit Skriptbefehlen benutzen. Doch auch hier ist es schwierig eine eindeutige Fehlermeldung auszugeben, denn es kann kaum beurteilt werden, ob es sich bei den Auswirkungen der betreffenden Befehle um kleine Spielereien und Zusatzelemente handelt, oder ob es Bereiche betrifft, die für die Funktionalität und die Zugänglichkeit der Seite entscheidend sind.

Anstatt also erneut unsichere Warnhinweise auszugeben, soll versucht werden, dem Benutzer des Evaluators eine Alternativansicht der Webseite zu erstellen. Dort kann er auf einen Blick oder durch ein paar kurze Aktionen selbst überprüfen, wie es um die Funktionalität der Seite bestellt ist, wenn Zusatztechnologien aus bestimmten Gründen nicht verfügbar sind oder nicht wahrgenommen werden können.

Der implementierte StructureTest repräsentiert daher keinen Test im engeren Sinne, sondern er ruft den TextViewDumper (ein HtmlVisitor) auf, der die analysierte Webseite von allen optionalen Elementen „säubert“ und das resultierende Grundgerüst in einer eigenen Ansicht ausgibt. Dabei werden folgende Aktionen durchgeführt:

- Die Tagblöcke <script>, <style>, <applet> und <object> werden komplett aus dem Dokument entfernt.
- Die leeren Tags <link> und <basefont> werden komplett aus dem Dokument entfernt.
- <img> Tags werden durch den Inhalt ihres alt-Attributs ersetzt.
- Aus dem <body> Tag werden die Attribute alink, link, vlink, text und onload entfernt.
- Aus <a> und <area> Tags werden href-Attribute entfernt, wenn diese ein Javascript als Verweisziel haben.
- Aus allen anderen Tags werden die Attribute style, color, bgcolor, background, face, size und sämtliche Eventhandler entfernt.

Die resultierende Textansicht der Seite öffnet sich in einem neuen Fenster und kann dann von dem Benutzer direkt mit der Originalseite verglichen werden, um festzustellen ob nach wie vor alle Bereiche sichtbar, erreichbar und logisch angeordnet sind, ob Informationen verloren gegangen sind, die nur über Bilder oder Farbinformationen dargestellt wurden und so weiter.

7 Evaluierung

In diesem Kapitel wird versucht, den Nutzen der unter „5 Umsetzung“ und „6 Implementierung“ beschriebenen Verfahren zu evaluieren. Dazu beschreiben wir zuerst eine allgemeine Methode zum Vergleichen von Accessibility-Evaluatoren und werden danach passende Teile davon auf den erweiterten Prototyp dieser Arbeit anwenden und die Ergebnisse präsentieren.

7.1 Methode

Giorgio Brajnik ist ein Professor an der Universität von Udine (Italien), wissenschaftlicher Berater von UsableNet Inc. und einer der Entwickler der LIFT Machine, einem kostenpflichtigen, serverseitigen System für Web-Qualitätsprüfung (Accessibility, Usability) [46]. Im März 2004 hat er in einem Paper eine Methode zum Vergleich und zur Effektivitätsbeurteilung von Accessibility-Evaluatoren vorgestellt.

Demnach hängt die Effektivität von solchen Tools grundsätzlich von drei Parametern ab:

1. Vollständigkeit
2. Korrektheit
3. Spezifität

Der Grad der Vollständigkeit gibt an, wie viele Probleme, die tatsächlich auf einer Webseite vorhanden sind, auch wirklich von dem Evaluator gefunden und entsprechend markiert werden. Es geht also darum zu verhindern, dass echte Fehler übersehen werden („false negatives“). Da die Vollständigkeit bezüglich dem Auffinden von „echten“ Probleme voraussetzt, dass man alle möglichen Probleme schon im Vorhinein kennt und charakterisieren kann, definiert Brajnik Vollständigkeit als Grad der Konformität zu den Richtlinien des W3C (WCAG 1.0).

Der Anteil der Probleme, die von einem Evaluator angezeigt werden und die auch tatsächliche Probleme darstellen, drückt die Korrektheit des Werkzeugs aus. Je geringer die Korrektheit ist, desto höher ist folglich die Anzahl an Fehltreffern („false positives“), die von dem Tool ausgegeben werden. Da im Accessibility-Bereich sehr viele Richtlinien mit der (individuellen) Wahrnehmung oder Interpretation von Inhalten zu tun haben, können Fehltreffer nie komplett ausgeschlossen werden. In diesem Fall kann der Evaluator den Benutzer mit weiteren Informationen und Entscheidungshilfen unterstützen.

Die Spezifität gibt schließlich an, wie detailliert das Prüfprogramm Fehler finden und beschreiben kann. Je spezifischer ein Fehler eingegrenzt werden kann, umso exakter kann auch die entsprechende Fehlerbehandlung erfolgen. Die vorliegende Vergleichsmethode nimmt dabei vereinfachend an, dass ein Tool umso effektiver ist, je spezifischer es ist [3].

Eine Beurteilung des Prototyps dieser Arbeit kann natürlich nicht all diese Kriterien berücksichtigen, da WUSAB kein vollständiges Regelwerk überprüft, sondern immer nur Verbesserungsmöglichkeiten in isolierten Bereichen aufzeigt. Eine Beurteilung wird sich deshalb hauptsächlich auf die (verbesserte) Korrektheit dieser Teilgebiete im Vergleich zu bestehenden Accessibility-Evaluatoren beziehen.

7.2 Durchführung und Ergebnisse

Um die Prüfsoftware zu testen, wurden einerseits Dummy-Webseiten aus den HTML-Testdateien des W3C zusammengestellt sowie einige Seiten mit realen Inhalten aus dem Web zur Prüfung ausgewählt. Diese mussten zuvor manuell auf etwaige Probleme oder bestimmte Merkmale untersucht werden, damit sie anschließend mit den Ergebnissen der Evaluatoren verglichen werden konnten.

7.2.1 Verbesserungen durch die Bildklassifizierung

Das Testfile bestand hier aus acht verschiedenen Bildern (dekorativ/normal, groß/klein), wovon vier einen korrekten ALT-Text hatten und die anderen vier einen fehlerhaften. Überprüft wurden die Evaluatoren WebXACT, WAVE 3.5, ATRC Accessibility Checker und der hier vorgestellte Prototyp.

Eines der Bilder hatte gar kein „alt"-Attribut gesetzt und dieser einfache Fall wurde auch von allen Tools richtig als Fehler erkannt.

WebXACT gab für alle anderen Bilder dann noch vier Warnungen aus, dass eventuell ausführlichere Beschreibungen über ein „longdesc"-Attribut nötig sein könnten. Ansonsten wurden weder für kleine noch für große Bilder mit leerem ALT-Text Fehlermeldungen ausgegeben. Leerer ALT-Text scheint hier also pauschal als richtig gewertet zu werden. Ebenso alle Bildgrößen, die irgendetwas im ALT-Text eingetragen haben.

WAVE 3.5 identifizierte neben dem einen Fehler weitere sieben so genannte „Accessibility-Features" und wertete damit alle anderen Grafiken und deren Alternativtexte als korrekt.

Der Checker von ATRC meldete den obligatorischen Fehler und drei allgemeine Warnungen, dass die Grafiken Text beinhalten könnten, der nicht im ALT-Text enthalten ist. Außerdem gibt das Tool von ATRC Warnungen aus, für alle Bilder, die größer als 25 x 25 Pixel sind und einen leeren ALT-Text haben. Sind solche Bilder kleiner als 25 x 25 Pixel, werden sie als korrekt (keine Fehlermeldung, keine Warnung) gewertet. Dies gilt allerdings auch für Bilder dieser Größe, die etwas im „alt"-Attribut stehen haben, möglicherweise aber trotzdem dekorativ sind.

Der WUSAB Prototyp markiert im Zwischenschritt der Auswertung zwei der Grafiken aufgrund ihrer Größe als dekorativ. Dies wird dem Benutzer klar ersichtlich angezeigt, so dass dieser mit einigen schnellen Klicks eventuelle Fehleinschätzungen des Programms beheben kann. Auf diese Art können auch die beiden größeren Bilder, die dekorativen Inhalt darstellen sollen, schnell händisch nachklassifiziert werden. Danach kann WUSAB für alle acht Bilder eindeutige Entscheidungen treffen. Natürlich kommt dieser Effekt semi-automatisch und mit Hilfe des Benutzers zu Stande, allerdings kann man durch das Einteilen einiger weniger Bilder in eine Klasse, eine ungleich höhere Anzahl an Warnungen und manuellen Nachprüfungen umgehen. Meistens werden pro Bild, das nicht eindeutig beurteilt werden kann, ja gleich mehrere Warnungen und „Vermutungen" aktiv, so dass sich der Aufwand multipliziert. Nur im ATRC Checker kann man während des Abarbeitens der Warnhinweise mit Hilfe von „decisions" auch überflüssige Zweige von potentiellen und möglichen Probleme überspringen.

7.2.2 Korrektheit der Textparser

Zur Prüfung der Textparser der Readability-Tools wurden fünf kleine Webseiten verwendet. Diese enthielten keinen dynamischen Content und wurden auf einem eigenen Server zwischengespeichert, um allen Tools dieselbe Eingabe zu geben. Vier Webseiten wurden mehr oder weniger zufällig ausgewählt (z.B. die Startseiten der jeweiligen Tools) und eine fünfte wurde eigens präpariert, damit beispielsweise <code> oder <script> Blöcke im Quellcode vorkommen.

Die Korrektheit wurde dabei aus der durchschnittlichen, prozentmäßigen Abweichung von den manuell gezählten Originalwerten der jeweiligen Seite berechnet. Anzumerken ist, dass die getesteten Webseiten nur sehr kleine Textausschnitte enthielten (200 bis 300 Wörter), da diese manuell überprüft und durchgezählt werden mussten. Aufgrund des geringen Umfangs kann sich ein Parse-Problem in einem größeren Bereich natürlich sehr stark auf die prozentmäßige Abweichung auswirken. Es wurde allerdings versucht, solche Bereiche der Seitengröße entsprechend klein zu halten. Tabelle 7.1 unten zeigt die Ergebnisse der Analyse. Man sieht, dass der erweiterte WUSAB-Prototyp sowohl bei der Wort- als auch bei der Satzzählung die genauesten Ergebnisse liefert.

Name des Tools	Korrektheit der gefundenen Wortanzahl	Korrektheit der gefundenen Satzanzahl	Sätze sind verzerrt (zu lang/zu kurz)
TxReadability	95,1 %	-	-
readability.info	92,9 %	66 %	ja
WUSAB Prototyp	98 %	93,9 %	nein

Tabelle 7.1: Korrektheit der Textparser von verschiedenen Readability-Evaluatoren

7.2.3 Weitere Anmerkungen

Die anderen in dieser Arbeit umgesetzten Bereiche konnten nicht direkt verglichen beziehungsweise evaluiert werden. Bei der Textansicht handelt es sich um eine Vorschau, die den Benutzer bei seinen Entscheidungen unterstützen soll, die aber selbst keine Bewertung vornimmt, welche mit der oben vorgestellten Methode untersucht werden könnte. Ob so eine Voransicht die Identifikation oder Behebung von Zugänglichkeitsproblemen wirklich beschleunigt und wie groß der zeitliche Gewinn ist, müsste mit einer weiterführenden Untersuchung unter Webautoren allgemein bestätigt beziehungsweise widerlegt werden. Unsere subjektive Einschätzung beim Testen des Prototyps zeigte dennoch eindeutige Vorteile einer solchen Visualisierung gegenüber allgemein gehaltenen Warnmeldungen, die ohne weitere Hilfe händisch (im Code?) überprüft werden sollen.

Auf dem Gebiet der Farbenkontrastanalyse gab es bei den hier betrachteten Accessibility-Evaluatoren kein entsprechendes Gegenstück. In einer Testseite, deren Farbkontraste händisch extrem verschlechtert wurden, konnte keines der untersuchten Tools Fehler entdecken. Der WUSAB-Prototyp hat diese Textpassagen hingegen alle richtig identifiziert und mit dem jeweiligen Farbkontrastverhältnis als Fehler ausgegeben. Die in sich geschlossene Korrektheit der Kontrastberechnung wurde auch bereits während der Umsetzung laufend an den unterschiedlichsten Seiten im Web getestet, manuell nachkontrolliert und verbessert. Teilweise aufgetretene Probleme resultieren häufig schon aus unsauberen Markup- oder Style-Definitionen der untersuchten Webseiten, die zum Beispiel den verwendeten CSS-Parser manchmal in die Knie gezwungen haben.

Bei den Readability-Kennzahlen wurde bereits mit der Analyse des Textparsers gezeigt, dass Verbesserungen in diesem Bereich möglich sind und die Formeln auch im Web zielgerichteter eingesetzt werden können als bisher. Ein direkter Vergleich von Ergebnissen der implementierten Formeln mit anderen Tools macht keinen Sinn, schon allein wegen unterschiedlichen Algo-

rithmen zur Silbenzählung oder zur Textauswahl. Ob die hier vorgestellten Verfahren zu einer genaueren Readability-Bewertung der untersuchten Webtexte führen, könnte nur durch weiterführende Studien im Linguistikbereich bestätigt werden und hängt schlussendlich von den zugrunde liegenden Formeln und deren Korrektheit ab.

8 Schlussfolgerung und Ausblick

Accessibility-Themen haben im World Wide Web eine große Bedeutung. Um ein umfassendes Bewusstsein für Barrierefreiheit und die Akzeptanz von entsprechenden Regulierungen zu erreichen, müssen diese Maßnahmen auf verständliche Art und Weise an die Menschen gebracht werden. Und es muss möglich sein, die Zugänglichkeit von Webseiten jederzeit schnell und unkompliziert zu überprüfen, damit Problembereiche klar identifiziert und verbessert werden können.

Das Ziel dieser Arbeit war es, nach Möglichkeiten zu suchen, wie man aktuelle Accessibility-Evaluatoren verbessern und dadurch nützlicher machen kann. Aus diesem Grund wurden einleitend aktuelle Entwicklungen im Bereich der Accessibility-Regelwerke untersucht und verschiedene (nationale) Umsetzungen miteinander verglichen. Es hat sich herausgestellt, dass es an Einheitlichkeit fehlt. Die Empfehlungen des W3C haben keinen verpflichtenden Charakter und werden deshalb in nationalen Gesetzen häufig angepasst. Außerdem werden die Richtlinienkataloge zunehmend komplexer und abstrakter, was es für einfache Webautoren schwierig macht, den Überblick zu behalten und die Regeln unkompliziert umzusetzen. Accessibility-Evaluatoren, deren Aufgabe es ist, bei der korrekten Umsetzung von Zugänglichkeitsregeln zu helfen, haben ein generelles Problem: Nur sehr wenige dieser Regeln können wirklich automatisiert geprüft werden. Meistens muss der Benutzer den Grossteil der Arbeit manuell erledigen. Jedoch bestehen durchaus Möglichkeiten, Teile dieser Arbeit zu automatisieren oder zu beschleunigen. Zum Beispiel durch die Einführung eines semi-automatischen Zwischenschrittes, in dem der Benutzer den Kontext beziehungsweise die Bedeutung von Inhalten näher spezifiziert, wodurch spätere Analysen beschleunigt werden können.

Im darauf folgenden Kapitel wurde speziell die Readability von Texten betrachtet, da es sich hierbei um einen der Punkte handelt, die für automatische Evaluatoren kaum einschätzbar sind. Bekannte Readability-Formeln stützen sich auf strukturelle Eigenschaften von Texten wie die durchschnittliche Wort- oder Satzlänge und sind deshalb zumeist nicht direkt für die Anwendung im Web geeignet. Dort ist es für Textinhalte wichtiger, durch Überschriften, Aufzählungen und Grafiken schnell erfassbar und überblickbar zu sein, anstatt eine klar strukturierte Satzform zu haben. Ein weiterer Kritikpunkt an diesen Formeln ist, dass sie dem tatsächlichen Inhalt der Texte keine Bedeutung schenken. Es wurden daher Möglichkeiten untersucht, wie man aktuell vorhandene Formeln exakter für Webzwecke verwenden kann und ob dabei auch das Vokabular berücksichtigt werden kann. Zusätzlich wurden Überlegungen angestellt, wie man die - für die Lesbarkeit ebenfalls wichtigen - Kontrastverhältnisse einer Webseite in die Überprüfung einfließen lassen kann.

Während der Umsetzung und Implementierung wurden schließlich drei Möglichkeiten vorgestellt, wie man semi-automatisch Metainformationen zu den Inhalten einer Webseite hinzufügen kann. Zuerst kann man versuchen, diese aus dem Kontext der Elemente einer Seite zu generieren. Diese Vorauswahl wird dann in einem Zwischenschritt der Evaluierung dem Benutzer präsentiert und dieser kann nun gegebenenfalls weitere Änderungen daran vornehmen (Möglichkeit 2). Zuletzt besteht noch die Möglichkeit, dass solche semantischen Informationen bereits während der Seitenerstellung in den Quellcode integriert wurden und von dem Evaluator aufgegrif-

fen werden können. Die Evaluierung dieser Methoden hat gezeigt, dass so mit relativ wenig Zusatzaufwand durch diese Klassifizierung von Objekten eine ungleich höhere Anzahl an manuellen Nachprüfungen eliminiert werden kann.

Es wurde ein Parser implementiert, der für Readability-Formeln, die auf die Satzform der Eingabetexte angewiesen sind, nur passende Bereiche zur Analyse auswählt, um das Ergebnis so genau wie möglich zu machen. Zusätzlich werden alle relevanten Wörter der Seite mit verschiedenen Wortschatzlisten verglichen und eine Verteilung des Textes auf diese berechnet. Dadurch wird eine inhaltliche Komponente zu den strukturellen Kennzahlen hinzugefügt.

Schließlich wurde eine Farbenkontrastanalyse vorgestellt, die mit Hilfe eines Stacks beim Durchlaufen des Dokumentmodells arbeitet und so Bereiche mit zu geringen Farbunterschieden identifiziert und an den Benutzer meldet.

In Summe wurde also gezeigt, dass in vielen Bereichen aktueller Prüfsoftware noch Verbesserungen möglich sind. Trotzdem muss man realistisch bleiben und beachten, dass gute oder schlechte Accessibility viel mit individueller Wahrnehmung zu tun hat und nur sehr schwer gänzlich automatisiert und ohne menschliche Hilfe beurteilt werden kann. Evaluatoren können den menschlichen Prüfer dabei aber unterstützen und ihm noch mehr und bessere Entscheidungshilfen und Visualisierungen bieten, damit Probleme schnell lokalisiert und behoben werden können.

Zukünftige Arbeiten am Prototyp werden die Integration der Parserkomponenten (HTML und CSS) beinhalten, um eine Anwendung zu schaffen, die in der Lage ist, die analysierten Webseiten selbstständig zu rendern. Außerdem ist eine Vervollständigung der Testfälle anzustreben, um irgendwann komplette Analysen gemäß WCAG zu unterstützen. Dabei stellt sich die Frage, ob die hier vorgestellten, prototypischen Verfahren nicht sowieso in ein Open-Source-Tool wie den ATRC Accessibility Checker transformiert werden könnten.

Weiters müssen auch noch ausführlichere Untersuchungen und Studien angestellt werden, die beispielsweise den Nutzen der strukturellen Voransicht oder die Korrektheit der Readability- und Wortschatzergebnisse exakt feststellen. Erst dann kann man entscheiden, wie weitere Schritte auf diesem Gebiet aussehen könnten.

Auch die Entwicklung, die das Semantic Web nehmen wird, ist für die Weiterentwicklung der Prüfsoftware sehr interessant, da man in Zukunft wahrscheinlich immer mehr Metainformationen direkt aus den Webseiten extrahieren kann und sich die Möglichkeiten der Automatisierung dadurch weiter verbessern.

Anhang A – Ergänzende Codebeispiele

```
<check id="1" key="img" confidence="high" status="5">
  <name>All <code>img</code> elements have …</name>
  <error>…</error>
  <description>…</description>
  <rationale>…</rationale>
  <repair>
    <how-to>Add an <code>alt</code> attribute …</how-to>
    <example>…</example>
  </repair>
  <test>
    <procedure>
      <step>Check this…</step>
      <step>Check that…</step>
    </procedure>
    <expected>
      <step>This is ok…</step>
      <step>That is ok…</step>
    </expected>
    <fail>
      <step>Repair this…</step>
      <step>Repair that…</step>
    </fail>
    <pass>
      <next-check number="3" />
      <next-check number="6" />
      <next-check number="16" />
    </pass>
  </test>
  <machine>
    <trigger element="img" />
    <function call="attribute-missing" node="." value="alt" />
    <xquery><![CDATA[ //img[not(@alt)] ]]></xquery>
  </machine>
  <test-file type="positive" src="1-1.html">
    <name><code>img</code> element without an <code>alt</code>
attribute.</name>
  </test-file>
  <test-file type="negative" src="1-2.html">
    <name><code>img</code> element with an <code>alt</code> at-
tribute.</name>
  </test-file>
</check>
```

Auflistung A.1: ATRC Accessibility Checker – Ein Testfall in der Datei checks.xml

Anhang B – Inhalt der beiliegenden CD

Inhaltsverzeichnis als Text-Datei
inhalt.txt

Ordner „dokumente“

dokumente/DA_Andreas_Singer.pdf	Diplomarbeit als PDF-Dokument
dokumente/DA_Andreas_Singer.doc	Diplomarbeit als Word-Dokument
dokumente/DA_Zwischenbericht.ppt	Powerpoint-Folien der Zwischenpräsentation

Ordner „wusab“

wusab/JavaSource/	Quelltext der Servletklassen
wuasb/WebContent/	Quelltext der JSP-Seiten und Scripte
wusab/doc/	Javadoc-Verzeichnis

Ordner „quellen“

quellen/pdf/	Verwendete Quellen, die als PDF verfügbar waren
quellen/web/	Verwendete Web-Quellen

Ordner „framework“

framework/	Software, die für die Diplomarbeit verwendet wurde

- Tomcat Application Server 5.5.17, Windows Installer
- JDK 1.5.0_08, Windows Installer
- Apache Ant 1.6.5
- JavaCC 4.0
- Eclipse SDK 3.2 for Win

Referenzen

[1] Apache Software Foundation. Apache Tomcat. Abgerufen am 10. November 2006. http://tomcat.apache.org/

[2] Atterer Richard, Schmidt Albrecht. Adding Usability to Web Engineering Models and Tools. In: Proceedings of the Fifth International Conference on Web Engineering (ICWE2005). Sydney, Australia. Seiten 36-41. Springer LCNS 3579. Juli 2005

[3] Brajnik Giorgio. Comparing accessibility evaluation tools: a method for tool effectiveness. In: Universal Access in the Information Society, 3 (3-4), Springer Verlag. Oktober 2004.

[4] Bundesministerium der Justiz Deutschland. BITV, Anzuwendende Standards. Juli 2002. Abgerufen am 07. Oktober 2006. http://www.gesetze-im-internet.de/bitv/__3.html

[5] Bundesministerium der Justiz Deutschland. Verordnung zur Schaffung barrierefreier Informationstechnik nach dem Behindertengleichstellungsgesetz. Juli 2002. Abgerufen am 07. Oktober 2006. http://www.gesetze-im-internet.de/bitv/

[6] Clark Joe. A List Apart Article: To Hell with WCAG 2. 23. Mai 2006. Abgerufen am 15. Oktober 2006. http://alistapart.com/articles/tohellwithwcag2/

[7] Cobb Tom. Research Uses of VocabProfile. Mai 2006. Abgerufen am 14. November 2006. http://www.lextutor.ca/vp/research.html

[8] Coleman M., Liau T. L. A computer readability formula designed for machine scoring. In: Journal of Applied Psychology, Vol. 60. 1975.

[9] Coxhead Averil. The Academic Word List. 2000. Abgerufen am 14. November 2006. http://www.vuw.ac.nz/lals/research/awl/awlinfo.html

[10] Dawson Carl. How useful are accessibility evaluation tools? Article at Accessites.org. September 2006. Abgerufen am 15. Oktober 2006. http://accessites.org/gbcms_xml/news_page.php?id=21#n21

[11] DuBay William H. The Principles of Readability. 25. August 2004.

[12] DuBay William. Plain Language at Work Newsletter. 23. März 2004. Abgerufen am 10. Oktober 2006. http://www.impact-information.com/impactinfo/newsletter/plwork08.htm

[13] Electronic Commerce Info Net. Rasantes Internet-Wachstum in Europa. 2001. Abgerufen am 02. Oktober 2006. http://www.ecin.de/news/2001/05/14/02064/

[14] Gulli A., Signorini A. The Indexable Web is More than 11.5 billion pages. 2005.

[15] Hellbusch Jan. Barrierefreies Webdesign. Barrierefreiheit international: globales Netz bedeutet globale Anforderungen. 2005. Abgerufen am 15. Oktober 2006. http://www.barrierefreies-webdesign.de/richtlinien/global/index.html

[16] Howard Hughes Medical Institute. Color Blindness: More Prevalent Among Males. Abgerufen am 14. November 2006. http://www.hhmi.org/senses/b130.html

[17] Initiative "Keine Macht den Barrieren". Wie steht es um die Barrierefreiheit in Deutschland? 2005. Abgerufen am 03. Oktober 2006. http://www.kmdb.de/wissen.htm

[18] Interface Consult GmbH. Usability Forum: Accessibility. Abgerufen am 02. Oktober 2006. http://www.usability-forum.com/bereiche/accessibility.shtml

[19] IT Accessibility & Workforce Division. Section 508 Law. Abgerufen am 07. Oktober 2006. http://www.section508.gov/index.cfm?FuseAction=Content&ID=3

[20] IT Accessibility & Workforce Division. Section 508 Standards. Abgerufen am 07. Oktober 2006. http://www.section508.gov/index.cfm?FuseAction=Content&ID=12#Web

[21] IT Accessibility & Workforce Division. Summary of Section 508 Standards. Abgerufen am 07. Oktober 2006. http://www.section508.gov/index.cfm?FuseAction=Content&ID=11

[22] Marincu Carmen, McMullin Barry. A comperative assessment of Web accessibility and technical standard conformance in four EU states. First Monday, volume 9, number 7. Juli 2004. Abgerufen am 15. Oktober 2006. http://www.firstmonday.org/issues/issue9_7/marincu/

[23] Mayrhofer Ronald. Die Welt der Farben, die Farben und ihre Bedeutung. Abgerufen am 14. November 2006. http://www.lichtkreis.at/html/Welt_der_Farben/welt-der-farben.htm

[24] McLaughlin Harry G. SMOG Grading – a New Readability Formula. In: Journal of Reading, 12 (8) 639-646. 1969

[25] McLaughlin Harry. SMOG – Simple Measure of Gobbledygook. Oktober 2006. Abgerufen am 14. November 2006. http://webpages.charter.net/ghal/SMOG.html

[26] Nielsen Jakob. Alertbox. How Users Read on the Web. 01. Oktober 1997. Abgerufen am 14. November 2006. http://www.useit.com/alertbox/9710a.html

[27] Nielsen Jakob. Alertbox. Usability 101: Introduction to Usability. 25. August 2003. Abgerufen am 02. Oktober 2006. http://www.useit.com/alertbox/20030825.html

[28] Nielsen Jakob. Jakob Nielsen Biography. Abgerufen am 03. Oktober 2006. http://www.useit.com/jakob/

[29] Österreichisches Bundeskanzleramt. E-Government. Abgerufen am 02. Oktober 2006. http://www.cio.gv.at/egovernment/

[30] Projekt „Barrierefrei informieren und kommunizieren". „BITV erfüllen – jetzt" Halbjahresbilanz Teil 1: Tests der Woche. September 2006. Abgerufen am 07. Oktober 2006. http://www.bik-online.info/test/bitv/bilanz_1.php

[31] Projekt „Barrierefrei informieren und kommunizieren“. „BITV erfüllen – jetzt“ Halbjahresbilanz Teil 2: 116 Vortests. September 2006. Abgerufen am 07. Oktober 2006. http://www.bik-online.info/test/bitv/bilanz_2.php

[32] Projekt „Barrierefrei informieren und kommunizieren“. Pressemitteilung zur Kampagne „BITV erfüllen – jetzt“. 15. Februar 2006. Abgerufen am 07. Oktober 2006. http://www.bik-online.info/test/bitv/pressemitteilung.php

[33] RFP Evaluation Centers. The Dale-Chall list of 3000 simple words. Abgerufen am 14. November 2006. http://www.rfp-templates.com/Dale-Chall-List-of-3000-Simple-Words.html

[34] Ridpath Chris, Treviranus Jutta. A-Prompt: Promoting the Habituation of Accessible Web Authoring. 2004. Abgerufen am 09. Oktober 2006. http://barrierfree.ca/tile/hci_formatted_edited1.html

[35] Singer Andreas. Digitales Lesen – Hat das Papier ausgedient? Dezember 2003.

[36] Smith E. A., Senter R. J. Automated readability index. AMRL-TR, 66-22, Wright-Patterson AFB, OH: Aerospace Medical Division. 1967.

[37] Statistisches Bundesamt Deutschland. Broschüre: Informationstechnologie in Unternehmen und Haushalten 2005.

[38] Statistisches Bundesamt Deutschland. Statistik der schwer behinderten Menschen 2003. Erschienen im März 2005.

[39] Sticht Thomas. Reading for Working: A Functional Literacy Anthology. Human Resources, Alexandria, Juni 1975

[40] Sun Microsystems. Java Servlet Technology. Abgerufen am 10. November 2006. http://java.sun.com/products/servlet/

[41] Sun Microsystems. JavaServer Pages Technology. Abgerufen am 10. November 2006. http://java.sun.com/products/jsp/

[42] Taylor Dave, Intuitive Systems. Readability.info: Readability Scores for Web Pages and MS Word files in a flash! Abgerufen am 14. November 2006. http://www.readability.info/

[43] TRACE Center, Adaptive Technology Ressource Center. HTML Tidy Accessibility Checker, Accessibility Checks. Dezember 2003. Abgerufen am 25. August 2006. http://www.aprompt.ca/Tidy/accessibilitychecks.html

[44] U.S. Department of Education. The Rehabilitation Act. 1998.

[45] University of Texas at Austin, Accessibility Institute. TxReadability Project. Abgerufen am 14. November 2006. http://www.utexas.edu/research/accessibility/research/summary/readability.html

[46] UsableNet Inc. Lift Machine – A Server system for Web Quality. Abgerufen am 10. November 2006. http://www.usablenet.com/products_services/lift_machine/lift_machine.html

[47] Wikimedia Foundation Inc. Wikipedia Artikel – Lesbarkeit, Definition. Abgerufen am 10. November 2006. http://de.wikipedia.org/wiki/Lesbarkeit

[48] Word Wide Web Consortium, Techniques for Accessibility Evaluation And Repair Tools. W3C Working Draft. 26. April 2000. Abgerufen am 10. November 2006. http://www.w3.org/TR/AERT

[49] World Wide Web Consortium, Introducing to WCAG 2.0 Working Draft Documents. 19. November 2004. Abgerufen am 15. Oktober 2006. http://www.w3.org/2004/09/wai-nav/intro/wcag20.html

[50] World Wide Web Consortium. About the W3C. 2006. Abgerufen am 03. Oktober 2006. http://www.w3.org/Consortium/

[51] World Wide Web Consortium. Authoring Tool Accessibility Guidelines 1.0. 03. Februar 2000. Abgerufen am 15. Oktober 2006. http://www.w3.org/TR/ATAG10/

[52] World Wide Web Consortium. EARL – the Evaluation And Report Language. Abgerufen am 09. Oktober 2006. http://www.w3.org/2001/03/earl/

[53] World Wide Web Consortium. Essential Components of Web Accessibility. November 2005. Abgerufen am 15. Oktober 2006. http://www.w3.org/WAI/intro/components

[54] World Wide Web Consortium. Requirements for WCAG 2.0. April 2006. Abgerufen am 04. Oktober 2006. http://www.w3.org/TR/wcag2-req/

[55] World Wide Web Consortium. Resource Description Framework (RDF). Abgerufen am 09. Oktober 2006. http://www.w3.org/RDF/

[56] World Wide Web Consortium. Semantic Web. Abgerufen am 01. November 2006. http://www.w3.org/2001/sw/

[57] World Wide Web Consortium. Techniques for WCAG 2.0. 27. April 2006. Abgerufen am 15. Oktober 2006. http://www.w3.org/TR/WCAG20-TECHS/

[58] World Wide Web Consortium. Understanding WCAG 2.0. 27. April 2006. Abgerufen am 14. November 2006. http://www.w3.org/TR/UNDERSTANDING-WCAG20/

[59] World Wide Web Consortium. User Agent Accessibility Guidelines 1.0. 17. Dezember 2002. Abgerufen am 15. Oktober 2006. http://www.w3.org/TR/WCAG20/

[60] World Wide Web Consortium. WAI, Introduction to Web Accessibility. September 2005. Abgerufen am 03. Oktober 2006. http://www.w3.org/WAI/intro/accessibility.php

[61] World Wide Web Consortium. Web Content Accessibility Guidelines 1.0. Mai 1999. Abgerufen am 03. Oktober 2006. http://www.w3.org/TR/WCAG10/

[62] World Wide Web Consortium. Web Content Accessibility Guidelines 2.0. April 2006. Abgerufen am 04. Oktober 2006. http://www.w3.org/TR/WCAG20/