UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

R2WEB: REPAIRING AND EVALUATING RICH WEB APPLICATIONS ACCESSIBILITY

Ana Sofia Batista Neves

DISSERTAÇÃO

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Sistemas de Informação

2013

UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

R2WEB: REPAIRING AND EVALUATING RICH WEB APPLICATIONS ACCESSIBILITY

Ana Sofia Batista Neves

DISSERTAÇÃO

Projecto orientado pelo Prof. Doutor Luis Manuel Pinto da Rocha Afonso Carriço

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Sistemas de Informação

2013

Acknowledgments

In the first place, i am most grateful to Prof. Luis Carriço for this opportunity, for the patience and for the support and guidance throughout this last year. To my coworker Nádia, i want to say that i recognize and thank, the valuable advices given to me from her experience.

Secondly it must be said to all those true friends who accompanied me until here, those who listened, those who cared, those who gave me support and taught me new things, those who got me away from my work when i really needed, i honestly appreciate you and what you did for me. Without you there would be no thesis at all. Thank you.

To my Mom, i want to show my appreciation for the unquantifiable support and for all the other unimaginable things only a Mom could give. I also thank you for the food, though it was me who cooked it.

To my Dad, i want to say thank you for the many things, only now, am i able to understand and value.

Finally, but not the most important, i want to say thank you to every single person that made this year unbearable. I most definitely have to say that if not for you, i would not have learned some important life lessons.

Resumo

A Web é hoje em dia a plataforma mais utilizadas para partilha de conhecimento e divulgação de serviços. A sua crescente importância nota-se aos mais diferentes níveis da nossa sociedade. Universidades, governos, empresas e outras entidades todas aproveitam da facilidade de chegar ao destinatário através da Web.

No entanto, o problema surge dado o facto de que nem todos aqueles, a quem este conhecimento e estes serviços se destinam, têm as mesmas capacidades, cognitivas ou físicas. Grupos sociais de indivíduos com dificuldades cognitivas e físicas acabam por ficar à margem destas potencialidades da Web.

Apesar de existirem já inúmeras tecnologias de ajuda para estas pessoas, como leitores de ecrã, etc., para que estas funcionem corretamente, é necessário que a informação seja disponibilizada adequadamente formatada. De forma a colmatar estas e outras situações existe a necessidade de normas cujo objectivo seja estandardizar conteúdos desenvolvidos para a Web.

A acessibilidade Web é uma preocupação da qual surgem os princípios de inclusão social. E graças à qual, inúmera documentação e tecnologias, têm surgido de forma a colmatar este problema. Entre elas, tecnologias automáticas de avaliação e até de reparação de Acessibilidade Web.

Contudo, muitas dessas tecnologias ainda não conseguem colmatar dificuldades que surgem do desenvolvimentos tecnológico da Web. Novas tecnologias, como Javascript, tornam os conteúdos Web em mais do que simples apresentações de texto. Conteúdos Web passam a ser dinâmicos, graças à execução de chamadas aos servidores, paralelas à apresentação das páginas Web. Estas interações com o servidor permitem a alteração dos conteúdo nas páginas Web.

A avaliação da acessibilidade destes novos conteúdos dinâmicos, não pode ser verificada utilizando os mesmos padrões usados até agora. Caso isto aconteça estes avaliadores de conteúdos estáticos, podem levar a conclusões incompletas que podem mesmo ser incorretas.

A ferramenta QualWeb é uma ferramenta de avaliação de Acessibilidade Web. No entanto, ao contrário das mencionadas anteriormente, recorrendo a tecnologias recentes, esta ferramenta permite avaliar aplicações Web dinâmicas.

Avaliações de acessibilidade realizadas por esta ferramenta seguem normas internacionais WCAG 2.0 graças à implementação de um modulo de técnicas HTML. No entanto existe ainda espaço para expandir o âmbito desta ferramenta. As normas WCAG 2.0 englobam não apenas o HTML mas também um conjunto de outras tecnologias de entre as quais o CSS.

Sendo que, com o crescimento da Web e da quantidade de conteúdos disponíveis online, maior atenção tem sido atribuída à utilização de estilos que possibilitam aos developers desenvolver conteúdos mais apelativos. Por este motivo a sua utilização tem aumentado e os CSS são hoje em dia, considerados como uma das tecnologias mais importantes para a Web. Data esta importância, este trabalho pretende expandir o funcionamento de ferramenta QualWeb, introduzindo lhe novas capacidades anívels dos CSS. Para isso incluímos um modulo de avaliação e reparação de CSS. Esta avaliação, tal como a já existente avaliação de HTML, segue as técnicas CSS WCAG 2.0.

Não só nos preocupámos em manter a modularidade da ferramenta, mas também nos preocupámos em torna-la ainda mais flexível. As avaliações de HTML seguiam uma determinada forma de execução de comprometia a flexibilidade da apresentação de resultados ao developer. Foi por este motivo que decidimos também alterar a perspectiva de apresentação dos resultados,. Esperamos assim melhorar o workflow do developer bem como, a sua experiencia com a nossa ferramenta.

Considerando que a complexidade das página Web tem vindo a aumentar, o tempo que um developer poderá eventualmente despender, analisando os seus conteúdos, segundo padrões de acessibilidade, pode ser considerável. De forma a mantermos os developers interessados nestes temas, sem que considerem estas verificações cansativas, é importante tornar a sua experiencia tão simples quando possível.

é com este objectivo que surgem ferramentas de reparação de acessibilidade. No entanto, se já as avaliações de acessibilidade podem ser algo ambíguas, esta situação agrava-se na reparação. Reparar uma página Web requer um conhecimento relativamente da página bem como uma visão geral que muitas vezes as ferramentas automáticas não possuem. Por este motivo o que propomos neste trabalho são sugestões de reparação, apresentadas ao developer, de forma a que a ferramenta de forneça toda a informação necessária para reparar a acessibilidade dos seus conteúdos.

Neste trabalho procedemos ao desenvolvimento de um módulo de reparação para as técnicas CSS desenvolvidas. O objectivo é acima de tudo, mostrar ao developer que pode

tornar as suas páginas mais acessíveis, educando-o sobre como o fazer e despoletando o seu interesse explicando também porque o deve fazer.

De forma a que toda esta informação possa ser divulgada, é necessário melhorar a interface da ferramenta QualWeb, desenvolvida paralelamente a este projecto. Desenvolveu-se paralelamente a este trabalho uma interface para esta ferramenta, no entanto esta interface tem como objectivo apresentar uma versão anterior do QualWeb. De forma a conseguir disponibilizar todos os conteúdos desenvolvidos foi necessário melhorar essa interface introduzindo lhe novas funcionalidades.

Todas estas componentes desenvolvidos, foram verificadas através de um corpus de teste. Para a avaliação desenvolvemos testes individuais, por técnica, e globais que simulavam a complexidade de uma página Web real. Para a reparação, procedemos a reparação de algumas das mais utilizadas páginas Web, o que nos permitiu fazer uma analise critica às nossas implementações. Por fim, de forma a validar a interface e as reparações, procedemos a testes com utilizadores.

Estes testes foram cruciais para o refinamento da nossa ferramenta, bem como para obtermos algumas conclusões interessantes no nosso trabalho.

No final, conseguimos, como pretendíamos, melhorar a ferramenta QualWeb, tornando a ainda mais competitiva nesta área. Conseguimos também validar toda a implementação realizada dos diferentes componentes e tirámos conclusões bastante interessantes. Nomeadamente que o conhecimento de ferramentas deste género, a nível de estudantes de universidade não é muito elevado e que estas ferramentas podem desempenhar um papel interessante para alterar esta situação.

Este trabalho ajudou também a pavimentar caminho para novas melhorias desta ferramenta de avaliação de Acessibilidade Web.

Palavras-chave: Web, Acessibilidade, Ferramenta de Avaliação, Ferramenta de Reparação, CSS

Abstract

QualWeb is an existing tool that evaluates Web pages' compliance with WCAG 2.0 HTML techniques. We wanted to enhance this tool's functions by adding a CSS module. For this we intend to create a CSS module, to extend the evaluation process, so that it evaluates HTML and stylesheets, but also to present the user solutions for every evaluation problem.

In this report we present the different development stages that our CSS evaluation and repair module went through. We describe how we interpreted the WCAG 2.0 techniques and how we turned them into code; how we processed every CSS rule in a .html document and how we established the connection between the different HTML elements and their CSS rule; how we build the repair process itself and how we managed to present it in an online user interface.

One of this stages includes an experimental study where we presented to a set of twenty Web developers, with different levels of HTML knowledge, our developments in order to obtain some feedback on how we were doing. In the corresponding chapter we describe our study and show how presenting repair suggestions for accessibility problems, helps users feel less lost and more self-assured when using our tool.

These findings will build ground for a future complete repair module for QualWeb and open way for further enhancements of this Web Accessibility evaluation tool.

Keywords: Web, Accessibility, Evaluation Tool, Repair Tool, CSS

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Being an open platform, the Web is a great place to share knowledge, for it allows people and organizations to easily make information available, in a variety of different formats, to other people and organizations. Examples of this are: Students that now benefit from e-learning; online shopping that has become a way to get home deliveries; Governments which have made several official services available on the Web.

The problem in this model is that not all people can perceive or understand information the same way. Different types of disabilities, ranging from low eyesight or even total blindness, to cognitive difficulties in reading, can affect the way a person accesses the content of a Web page.[1] Because our world is moving more and more online, we need to see that this content is accessible, so that we ensure that people with disabilities are not excluded from society.[2]

"As the Web community grows and its members diversify in their abilities and skills, it is crucial that the underlying technologies be appropriate to their specific needs. HTML has been designed to make Web pages more accessible to those with physical limitations.(...)"[3]

Web accessibility shows up as the practice, which has the purpose of making Web content available to everyone, regardless of possible limitations. It implies that Web developers and Web designers should make their products accessible to these people with disabilities. Though nowadays most people with disabilities have access to assistive technologies[4], for these to function correctly, content needs to be correctly organized.

The Web Accessibility Initiative (WAI) is one answer to this increasing demand for inclusive Web sites. As part of its work, the WAI published several guidelines, some of them known as the Web Content Accessibility Guidelines (WCAG) 1.0 and 2.0,[5].

Together with accessibility guidelines, automated accessibility evaluators have been appearing all over the Web. Their objective is to verify the accessibility of a Web page in an easy way, signaling problems and potential problems. Developers are expected to use these tools to scrutinize their Websites, instead of making verifications manually, so that they can confirm the accessibility of their Website without spending too much time. This should be done during the developing cycle and should be granted as much importance as usability testing.[6]

Awareness of the need for inclusive Web content and accessibility issues is increasing among developers and these tools for developer's aid, do not go unnoticed anymore[7]. However their use is still considered cumbersome. Issues such as "lack of time" and "limitations in technology" were the main limiting factors found in a study by Shari Trewin et all[7]. Also, establishing the importance of accessible Web pages to management, who seemingly show favour to pleasing sites instead, has proved difficult.[8].

1.2 Objectives

An automated evaluation tool runs specific accessibility guidelines on a HTML input, producing a error report as a result which will include a list of infringements identified. There are already several evaluation tools online developed by different countries and entities. One of these tools is QualWeb an HTML WCAG 2.0 technique based evaluation tool for RIA. As we will describe ahead in greater detail, QualWeb already presents an interesting breakthrough comparing to other evaluation tools.

When studying the automated tools for accessibility available nowadays, we come across some tools that go beyond informing users what incorrect thing they have done. There are some tools that introduce the concept of automated repair. Known as repair tools, they perform evaluations, but then go further, by correcting specific situations identified as problems. There are some repair tools available online and in the next chapter we will talk more about them. However these tools are still limited and as we will see, do not deal with specific guidelines.

With this work we propose to expand QualWeb with some new functionalities that can be summarized as the following projects objectives.

- Our first objective is to implement WCAG 2.0 CSS techniques, allowing QualWeb to perform evaluations of stylesheet accessibility. This will include some subtasks such as:
  - The gathering of all the CSS in a Web page.

  - Implementing the CSS techniques.

- The second objective is developing a repair module for QualWeb. This module will receive the evaluation results and will suggest specific repairs, for the developer to apply.
- During this work a Web interface was developed for QualWeb. However this interface was developed only with the current version of QualWeb in mind. The Third Objective of this project is to extend the QualWeb interface, in order to allow the presentation of both CSS evaluation results and the suggested repairs developed.
- It is necessary to develop several tests, with the objective of verifying the implementation of these components.

1.3 Contributions and Publications

During the development of this project, we made several changes to the functioning of the existing automated accessibility evaluation tool QualWeb. When we started, QualWeb was already an amazing tool for evaluating Rich Internet Applications, using phantom.js as a headless browser to obtain dynamic content. Still, this project enriched the tool by:

- Adding CSS WCAG 2.0 techniques to the evaluation process.
- Optimizing the running and triggering of both CSS and HTML techniques
- Developing a new module that gathers all CSS in the given Web page.
- Developing another module that creates the repairs to be suggested.
- Enhancing the meanwhile developed QualWeb interface, by adapting it to show the CSS results as well as the correspondent suggested repairs.

During this process, the following papers were published:

- Nádia Fernandes, Ana Sofia Batista, Daniel Costa, Carlos Duarte, and Luís Carriço. 2013. Three web accessibility evaluation perspectives for RIA. In Proceedings of the 10th International Cross Disciplinary Conference on Web Accessibility (W4A'13). ACM, New York, NY, USA, Article 12 , 9 pages.
- Daniel Costa, Nádia Fernandes, Sofia Batista, Carlos Duarte and Luís Carriço. 2013. Web Accessibility in Africa: a Study of Three African Domains. INTERACT'13

1.4 Planning

1.4.1 Tasks

1. Related Work Study

   Study and understanding of Web accessibility concepts as well as some work done in this area such as evaluation and repair tools. This will help delineate what we should do in order to make QualWeb even more competitive. It should accompany the entire project as concepts and tools should constantly be studied and used for comparisons.

2. Requirement Analysis

   Gathering of all the requirements for the different modules to be developed.

3. Writing the Preliminary Report

   Writing of a preliminary report, that summarizes the knowledge obtained in the previous steps, as well as work already finished and work to be done in the future.

4. System design

   Design the architecture, the modules and the interface as well as comparing QualWeb's architecture before and after this work.

5. Implementation

   (a) Gathering all CSS

   Developing CSS pre-processing algorithm for obtaining all types of CSS and organizing them into a single iterable structure.

   (b) Coding the CSS Techniques

   Implementing and including onto QualWeb's evaluation process, the CSS techniques in the WCAG 2.0 webpage.

   (c) Coding the Repairs

   Implementing and including the repair module onto the QualWeb execution process.

   (d) Coding Optimizations

   Taking advantage of the opportunity to review and optimize the QualWeb's execution process, after all the previous modules are added to QualWeb.

   (e) Coding the Interface Enhancements

   Preparing the interface for the user testing stage. Proceed to the enhancement of the QualWeb online Interface, by including all the necessary components, in order to present the CSS evaluation results and the repair suggestions.

6. User Testing and Result Analysis

   Establishing test tasks and gathering test individuals in order to evaluate the interface, the evaluations and matching repairs. After the tests this also includes proceeding with the analysis of the results obtained.

7. Improvement of the overall code

   Refining, after the user testing task, all previous implemented modules in order to adjust and repair problems identified during the tests.

8. Writing of the Final Report

   Writing the final report of the project. This task, just as the first one, must be constant during the whole duration of the project, although most of it will be done in the last couple of months of the schedule.

1.4.2 Project schedule

This project was planned to last from September until June and for this period all tasks, described in the previous section, were defined and scheduled. In figure 1.1 we introduce the initial planning for these tasks, including how long they were initially planned to last. Figure 1.2 on the other hand, presents the actual duration of each task.

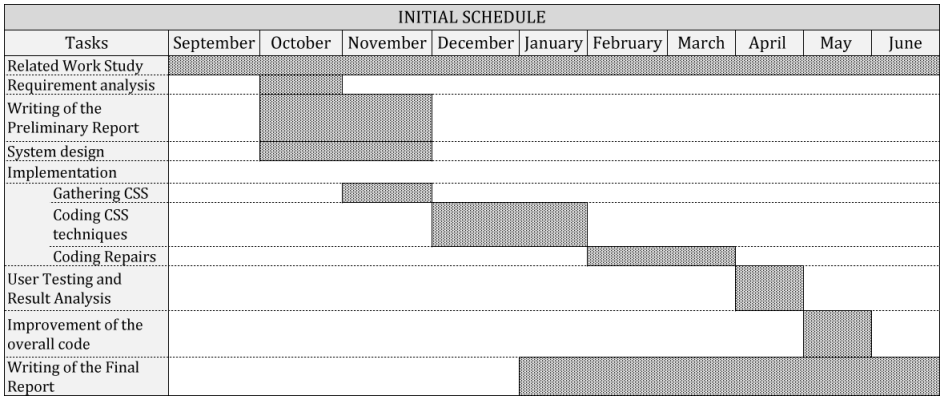

Figure 1.1: Mapping of the Initial Schedule

Comparing both schedules we can observe how some tasks seem to prolong beyond the period they were planned for. This happened as tasks in question, were not given due credit and turned out more challenging than was expected. We can also point out that new tasks were added to the revised schedule. These tasks were needed to improve the overall functioning and coherence of the QualWeb structure and as preparation for the User Testing.

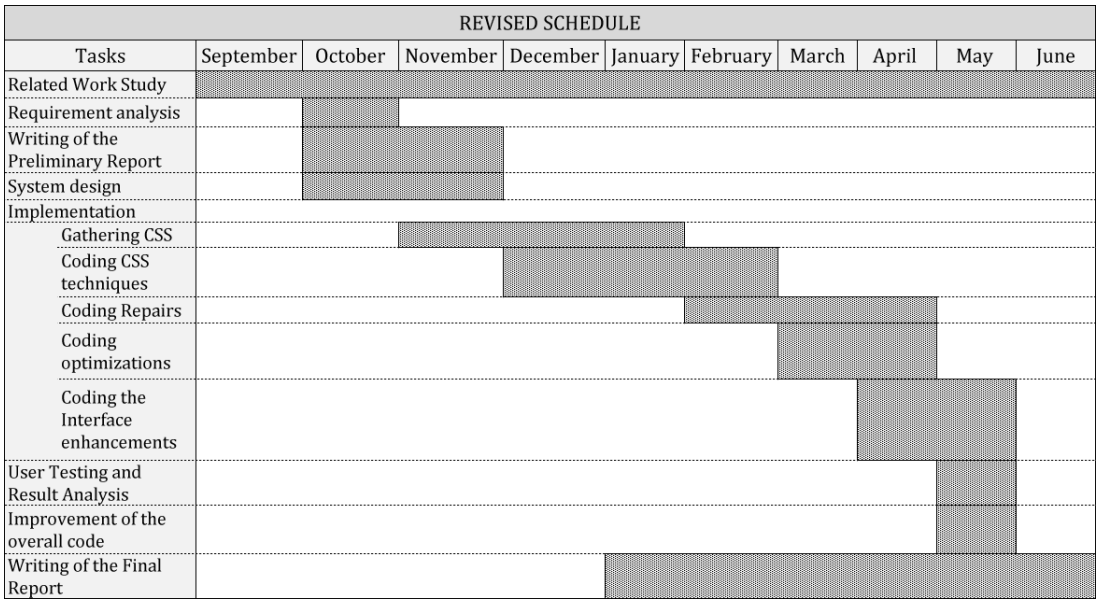

Figure 1.2: Mapping of the Revised Schedule

1.5 Document Structure

This document is organized into seven chapters.

- Chapter 2 - *State of the Art* - Presents a detailed study of the State of the Art of Web Accessibility and other related technologies.
- Chapter 3 - *Architecture* - Introduces the QualWeb evaluation tool and explains, in detail, all the modules that were added with this work.
- Chapter 4 - *Gathering the CSS* - Accounts for the process of retrieving and processing CSS
- Chapter 5 - *Evaluating the Web* - Explains all the different steps of the evaluation process.
- Chapter 6 - *Repairing the Web* - Details the entire repair process and the difficulties we faced during this development stage.
- Chapter 7 - *QualWeb Interface* - Reports the process of developing the enhancements of QualWeb's user interface. It also details the user testing method and the results we obtained.
- Chapter 8 - *Conclusions & Future Work* - Outlines our conclusions, lessons learnt during this project and in what ways this work can be continued.

Chapter 2

State of the Art

As the Web grows and increasingly becomes the best place to share information, not only more people want to obtain information, also more people want to distribute it. However because of this, we are now being confronted with a growing number of resources, developed by all kinds of people, with and without developer background. One example of this is public services that have now become available online. The problem is that people do not perceive things the same way, some have low eyesight some are blind, some cannot use a mouse or a keyboard.

So, in order to make sure this information is distributed in a way that can be perceived by all, Web developers, with experience or without, must be instructed of the best ways they should develop their contents.

2.1 Technologies

This section focuses on the technologies used during the development of this work. We focus on the Hyper Text Markup Language (HTML), Cascade Style Sheets (CSS) and Javascript since they play a major role in the developments of Web contents.

We also refer Browsers and Headless browsers since they are of great importance to our work. While Browsers are responsible for rendering content, which may or may not have accessibility problems, headless browsers are of great use for tools such as QualWeb.

2.1.1 HTML

Web contents are mainly developed in the Hyper Text Markup Language (HTML) with decorations done through Cascade Style Sheets (CSS). HTML is a language based on markup tags which describe the document structure, introducing areas such as headings,

paragraphs, links, inputs, and so on. Inside these tags plain text can be added to supplement content to the document structure. Tags are keywords surrounded by angle brackets <html>and that often come in pairs, start tags <html>and end tags </html>delimiting their content.

The structure of a HTML document is composed of at least three parts:

1. A doctype instruction for the Web browser containing what version of HTML the page is written in.

2. A header section (delimited by the HEAD element), instruct the browser where to find style sheets, provide meta information.

3. A body which contains the contents of an HTML document, such as text, hyperlinks, images, tables, lists, etc.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd" >
<HTML >
    <HEAD >
        <TITLE >My first HTML document </TITLE >
    </HEAD >
    <BODY >
        <P >Hello world! </P >
    </BODY >
</HTML >
```
[1]

2.1.2 CSS

In order to create greater aesthetics in Webpages, web developers have been using CSS to style elements in markup languages tags. This styling enhances presentation, turning contents in Web pages more than just static and plain information. The use of CSS to introduce these decorative elements has provided a way for developers to separate HTML contents form their decorations. This separation has allowed for several things, among which, one is to develop more compelling Web pages and other is to allow greater accessibility since users can adapt the presentation of content to their needs.

[1] example obtained from http://www.w3.org/TR/html401/struct/global.html

Types of CSS

There are different ways to introduce CSS specification in a Web page. CSS can be presented inline, internally and externally.

- Inline: As the value of the style attribute of the element:
  This is a paragraph.
- Internal: if it is between style tags, generally on the head section of the Web page

```
<head>
<style>
    hr {color:sienna;}
    p {margin-left:20px;}
    body background-image:url("images/back40.gif");
</style>
</head>
```

- External if it is introduced in the HTML structure with a <link >tag

```
<head >
<link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
```

CSS can also be introduced into a Web page inside scripts. When the page is loaded and these scripts executed, the new CSS are added to the already existing CSS. The place where these CSS are added to, depends only on the script that is executed.

CSS Rules

Figure 2.1 [2] shows that CSS rules follow a specific structure. This structure is divided into two parts, the selector and the declaration. Selectors are used to specify which tag the style applies to and declarations consists of the CSS properties and values.

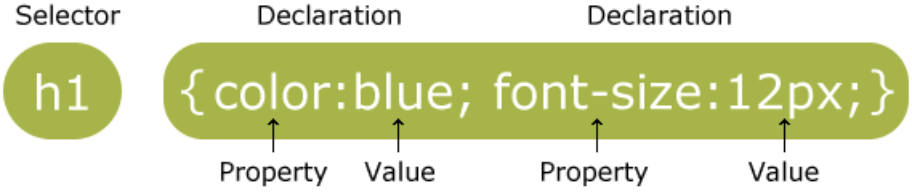

Figure 2.1: CSS structure example

[2] Obtained from http://www.w3schools.com/css/css_syntax.asp

Selectors

Selectors are the beginning of any CSS rule and are used for specifying to which tag the declaration applies to. They can be specified in three different ways.

- Elements of a specific type. (Figure 2.2 [3] for all headings of level 1 and level 2)

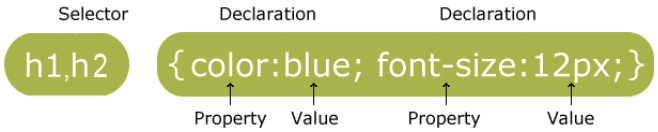

Figure 2.2: CSS double selector example

- Elements identified by attributes such as ID or Class (figures 2.3 and 2.4 [3]). For these CSS to be correctly applied to the HTML tag. The HTML tags must be declared with these attributes :
  - <h1 id="Hid">

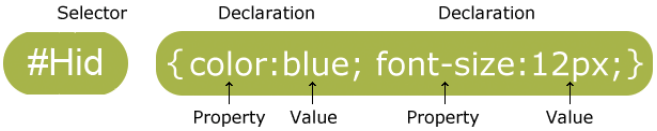

Figure 2.3: CSS ID selector example

  - <h1 class="Hclass">

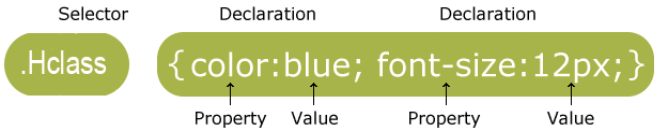

Figure 2.4: CSS Class selector example

- Elements placed relative to or nested within others. CSS in Figure 2.5[3] would be applied to any heading of type H1, inside any div, inside the element with the id "main".

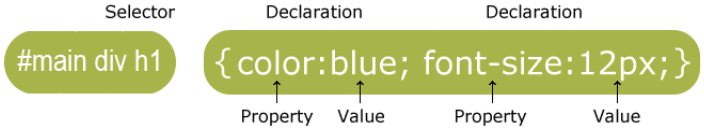

Figure 2.5: CSS Compund selector example

[3] Adapted from http://www.w3schools.com/css/css_syntax.asp

Pseudo-classes

Selectors can be enhanced further by adding other characteristics. These characteristics are called Pseudo-classes and are included in order to add special effects to the elements.

- a:link color:#FF0000; —>unvisited link
- a:visited color:#00FF00; —>visited link
- a:hover color:#FF00FF; —>mouse over link
- a:active color:#0000FF; —>selected link

Declarations

The declarations are CSS properties which are to be applied to the selector. CSS rules can have several declarations inside, as can be seen in figure 2.1. Tags of headings of level 1 will all inherit both declarations in that CSS rule.

Declarations are composed of properties and values. There are several CSS properties which can take several values. Although we do not specify all properties in this report, they can be found here http://www.w3schools.com/CSSref/default.asp.

Priorities

Since CSS for an element can be specified in different ways, priorities for rendering have been established.

1. Inline CSS takes priority over any other type;
2. CSS with specific selectors such as ID and Class, take priority over general CSS;
3. General CSS come last, and also depend on the order they are defined.

```
<style>
#someid {color: green};
p {color: blue;}
p {color: yellow;}
</style>
</head>

<body>
<p>This text would appear in Yellow</p>
<p id="someid">This text would appear in Green</p>
<p style="color:pink">This text would appear in Pink</p>
</body>
```

This text would appear in Yellow

This text would appear in Green

This text would appear in Pink

Figure 2.6: CSS Compund selector example

2.1.3 Browsers

Web browsers are applications used to locate, retrieve and present HTML Web pages, applications, JavaScript, other content hosted on Web servers and files in file systems. In order to locate these resources a Uniform Resource Identifier (URI) is used. This URI enables users to interact with the resources since it uniquely identifies a specific resource. Some of these resources are the HTML documents that may or may not be styled with CSS. Web browsers also allow users to do numerous tasks with these resources. For example, login and register, view and hear multimedia, print, send and receive email, among many others.

Retrieving Resources

To retrieve a resource a sequence of communication steps between the browser and the Web server occurs as can bee seen in figure 2.7[9].

- Web page refers to the resource identified by the URI. It defines the skeleton of the content that will be presented in the Web browser;
- Resources are complementary assets such as multimedia resourcers, stylesheets and scripts that are explicitly specified in the Web page's structure (within the proper HTML tags);
- AJAX request refers to the asynchronous communications done between the Web browser and the Web server. This kind of communication, transmitted without interfering with the display and behaviour of the Web page and allows web pages to became dynamic instead of just static displays of content, based only on CSS and HTML.

Browsers Rendering Engine

There are several HTML/CSS web browser rendering engines, depending on the browsers. For this report the reference Browsers will be Firefox which uses gecko, while Safari and Chrome use WebKit. Figure 2.8[4]

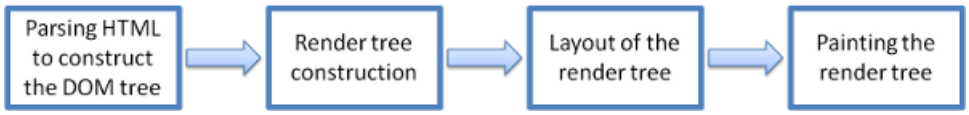

Figure 2.8: Browser Rendering Process

[4]All images obtained from http://taligarsiel.com/Projects/howbrowserswork1.htm#Parsing_general

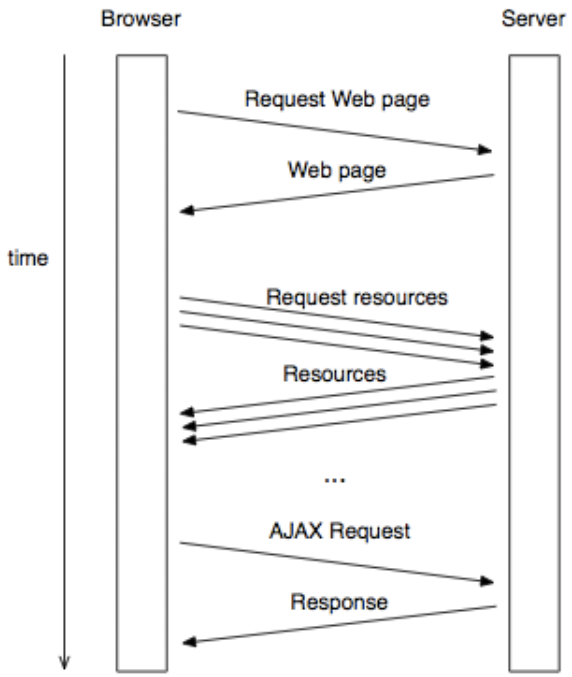

Figure 2.7: Web Browsing Resource Processing

1. The first step will be to start parsing the HTML document and turn the tags to DOM nodes in a tree called the ”content tree” and parse the styles.

2. The style information together with visual instructions in the HTML will be used to create another tree, the render tree. This structure contains rectangles, in the order to be displayed, with visual attributes like color and dimensions.

3. The layout process includes giving each node the exact coordinates where it should appear on the screen.

4. During the painting stage render trees will be traversed and each node painted.

This is a gradual process. The rendering engine will try to display contents, as soon as possible and it will not wait until all HTML is parsed before starting to build and layout the render tree. Parts of the content will be parsed and displayed, while the process continues with the rest of the contents that keeps coming from the network.

Figures 2.9 and 2.10 [4] illustrate the rendering process according to the different Browser rendering tool.

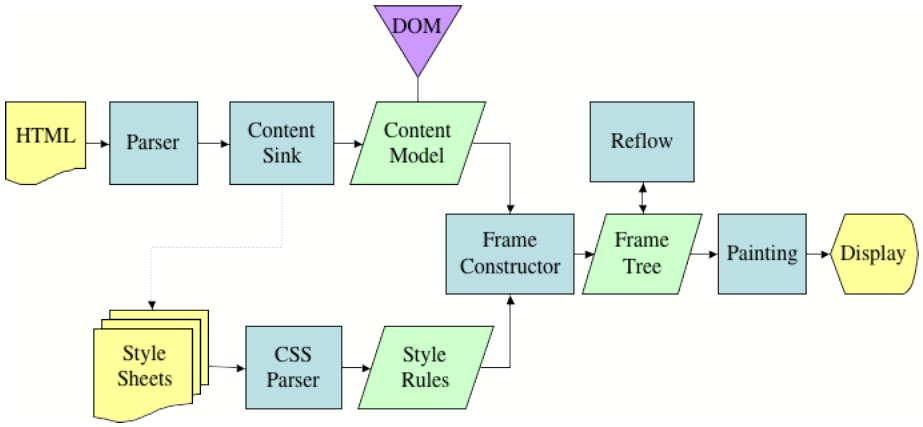

Figure 2.9: Steps to render an HTML page in Gecko

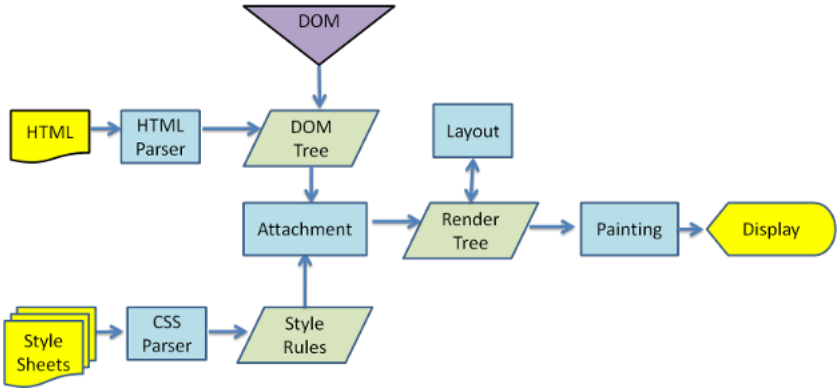

Figure 2.10: Steps to render an HTML page in WebKit

Parsing

The HTML resource obtained is written in a somewhat human understandable language. For it to be machine oriented it needs to be transformed into something a machine can understand and use. We call a parser something that does this.

For a parser to be correctly used, it requires that the language the document was written in, has a deterministic grammar, consisting of vocabulary and syntax rules such as CSS.

The DOM

HTML is not a language that can be parsed using a conventional parser since it allows you omit certain tags which are implicit, for example the end of certain tags.

The DOM is a cross-platform and language-independent interface that allows pro-

grams and scripts to dynamically access and update the content, structure and style of documents[5]. When parsed into a DOM the HTML is formatted into a tree structure, where tags and contents are kept in nodes and branches establish the connections between these tags and contents.

Parsing Style Sheets

Unlike HTML, CSS is a context free grammar and so can be parsed. Indeed we have shown already how CSS follow a specific grammar.

To parse CSS Webkit uses bottom up shift-reduce parser while Firefox uses a top down parser written manually. In both cases each file is parsed into a CSS object, containing selector and declaration objects. Figure 2.11[4] gives an example of how WebKit parses CSS.

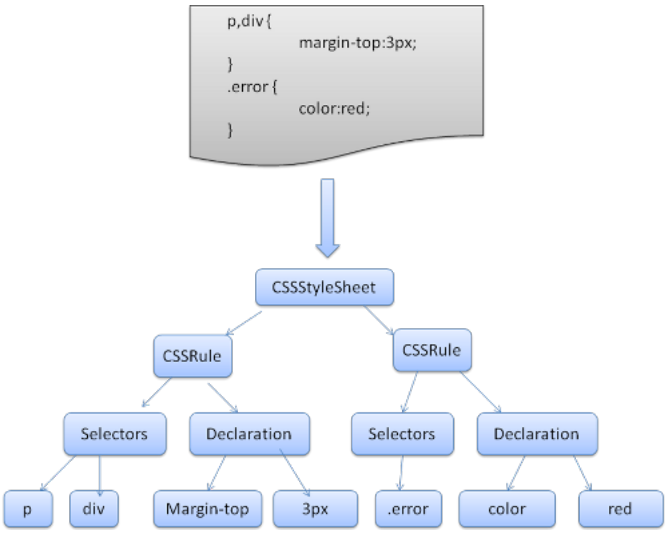

Figure 2.11: WebKit CSS parser

Render Tree and Frame Tree

These names both refer to the same thing, a structure with every element that is to be displayed as well as its placement. It contains visual elements in the order in which they will be displayed as a visual representation of the document. The purpose of this tree is to enable painting the contents in their correct order.

Non visual elements, will not be inserted. An example is the ”head” element and elements whose display attribute has value ”none”, since they will not appear in the tree.

[5]http://www.w3.org/DOM/

The objective of keeping a visual representation of the document is to enable styling the contents in their correct order.

Firefox calls the elements in the frame tree, ”frames”, while Webkit uses the term renderer tree and render objects. Each renderer consists of a rectangular area and contains geometric information like width, height and position.

Computing Style Sheets

Building the render tree requires calculating the visual properties of each element. This is done by obtaining and matching all the CSS properties of each element. However creating this matching between every CSS and elements can be a heavy computation process.

Firefox keeps two threes, one the context tree and the rule tree. The context tree keeps the structure of the visual elements as the DOM would. The rule tree, on the other hand, keeps all the style information organized in a hierarchy organized by elements, classes and ids.

To apply the CSS rules to elements in the context tree we just need to compute the paths from the context tree in the CSS rule tree. For example[6]:

Considering the Rules:

1. div {margin:5px;color:black}
2. .err {color:red}
3. .big {margin-top:3px}
4. div span {margin-bottom:4px}
5. #div1 {color:blue}
6. #div 2 {color:green}

Lets interpret the context and rule trees for this set of items:

In this image we can see that, for our better understanding, the context tree already has letters that refer to the rule tree. We can see how we can easily find the CSS that are chosen for each element from the rule tree. It is interesting to notice first, that for paragraph and body there are no CSS, second, that span tags inherit properties from above in the rule tree. Inherited properties are properties that unless defined by the element, are inherited from its parent. Thirdly span siblings can and will share the same CSS.

After being parsed both WebKit and Firefox, add the CSS rules to one of several hash maps, according to the selector. There are maps by id, by class name, by tag name and a general map for anything that does not fit into those categories. If the selector is an

[6]Adapted from http://taligarsiel.com/Projects/howbrowserswork1.htm

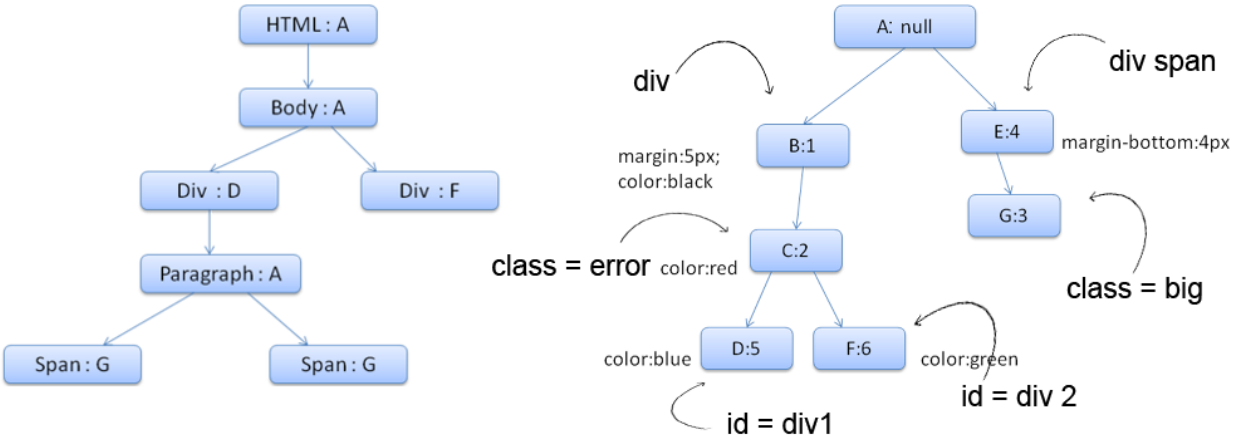

Figure 2.12: Context Tree (Left) and Rule Tree (Right)

id, the rule will be added to the id map, if it's a class it will be added to the class map etc. Manipulating CSS this way improves the matching process of rules.

As an example, the CSS: p.error {color:red}, will be inserted into the class map. Then in the HTML, when we have an instruction as: an error occurred , the Browser will first try to find rules for the p element in the class map.

Rendering Priority

CSS for a single element can be defined in several different style sheets and several times inside a style sheet. For this reason it is important to specify the order by which they will be applied. Several priorities have been established such as:

1. User important declarations signaled by !important
2. Author important declarations signaled by !important
3. Author normal declarations
4. User normal declarations
5. Browser declarations

2.1.4 Headless Browsers

So far we have seen that Browsers are used for people to access the Web and its resources. We have also seen how they process HTML and CSS in order to render pages. However, there are also tools, whose functioning is similar to that of a Browser, but still are not to be used by people. These machine oriented tools do not have a graphical user interface and are used to provide the content of web pages to other programs.

One example of their importance is the execution of JavaScript, as a normal browser would. Scripts in a Web page can be executed, in order to render the Web page as the user would visualize it. CSS specified inside scripts, can only be accessed after the execution of these scripts.

This is very important for this work, as an accessibility evaluator tool can perform more detailed evaluations by obtaining this version of the page.

Dynamic Web and States of a Web page

A Rich Internet application (RIA) is a Web content that has many of the characteristics of a desktop application. It can be delivered by a browser plug-in, extensive use of JavaScript and others. In RIAs new contents can be obtained using Javascript/AJAX, without refreshing or loading a new page. A user interaction (Figure 2.13[9]) can easily, through Javascript, modify visible elements without requesting data from a server. Alternatively an AJAX request to the server, can fully modify the presented content. In both cases a new version of a page will be available without changing the URI [10].

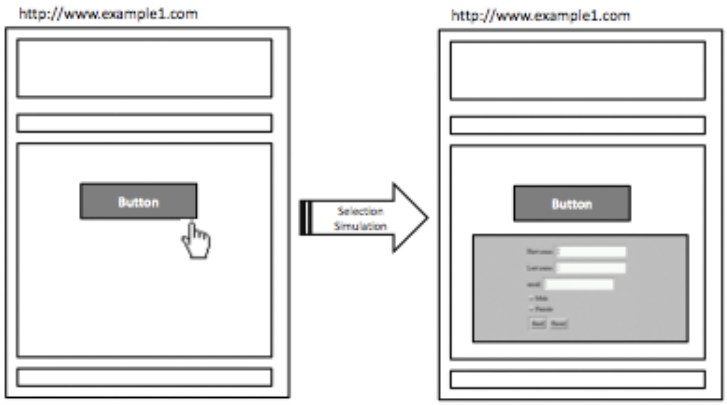

Figure 2.13: New state in dynamic applications

PhantomJs

In order to capture these RIA events, QualWeb takes advantage of the headless browser PhantomJS. A browser with WebKit engine and with a JavaScript API that gives access to Web pages from the command line while nothing gets displayed to the screen.

The usage of PhantomJs is crucial for the correct evaluation of a Web page. As evaluators obtain documents with written code, in HTML, CSS and JavaScript, for example, it obtains the documents as they were written but not as they are displayed to the user. This implies leaving many things unverified. For example, scripts that allow Web pages

to change their contents, will generate new HTML or CSS code to be inserted on the Web page without reloading it. This new code will not be evaluated since it is kept inside a script tag and not inside an HTML one, as they would be after the script's execution. This is heavily connected to what we mention before regarding Dynamic Web and States of a Web page.

2.1.5 NodeJS and JavaScript

NodeJS

NodeJs is a server side software tool for developing scalable internet applications, outside the Browser, specifically web servers. Tools are written on the server side in JavaScript using asynchronous I/O to allow scalability and reducing processing time. Also Node.js, already contains a built-in HTTP server library, which allows for running a web server without the use of external software, such as Apache. It also provides useful modules, for example, something that outputs a string on the console[7]. Node.js enables developers to create an entire web application in JavaScript, both server-side and client-side.

Javascript

NodeJS also provides an asynchronous environment where the developer can program using JavaScript. JavaScript is a programming language used on the Web by the majority of the Web pages and interpreted by all Browsers[11]. It allows Browsers to use client-side scripts to communicate asynchronously with the server, and alter displayed document content. Recently it has also become common for game development.

This work relates to Javascript in two ways. First, since it is used in the majority of Web pages and allows for the injection of new code, without reloading the Web page. This is the reason, Web evaluation tools need to take advantage of headless Browsers. As we mentioned before, pages evaluated when scripts have not been executed cannot, in reality, check Web pages as they are presented to the user. Which can cause several problems to escape detection. Second, this work will be developed inside the node environment and thus implemented in JavaScript language.

[7] http://www.nodebeginner.org/#javascript-and-nodejs

2.2 Web Accessibility

Through the process of developing Web contents, we are encouraged to think of the user we are developing for. However we tend to visualize users according to our own standards. As we are not all equal, we do not possess the same capabilities, nor the same characteristics and especially not the same limitations. This leads us to misunderstand others and consider specific things as irrelevant. Why does a blind programmer produce visually messy code?

Even if each individual is different and has different necessities, the fact is that he will want or need to use the things others are using too. One of these things will probably be Web. So with this in mind, the Web can keep on being even more creative, innovative and extravagant than it is, but still it must also not exclude anyone, Thought must go to the uncommon user of contents. Web Accessibility focuses on this, on the inclusion of each specific individual.

2.2.1 Accessibility Support

A specific group that more easily suffers from Web exclusion is that of people with disabilities. When Web applications and Web tools are badly designed, they can create barriers that exclude people from using the Web. With them in mind, several tools, called assistive technologies, have been already developed, with the purpose of enabling them to access contents on the Web.

Assistive technologies

Assistive technologies are devices or tools for people with disabilities. They promote greater independence by enabling them to perform activities they would not be able to do otherwise, or by allowing them to perform faster than they would normally. They provide enhancements or a different way of interacting with what people need, in order to accomplish a task. A few examples of assistive technologies are :

- Screen reader software can read out, using synthesized speech, what is being displayed on the monitor or everything that is happening on the computer.
- Braille terminals consist of a Refreshable Braille display which renders text as Braille characters, by means of raising pegs through holes in a flat surface.
- Screen magnification software enlarges what is displayed on the computer monitor, making it easier to read for vision impaired users.

- Speech recognition software are tools that can speak commands to the computer, or turn dictation into grammatically correct text, for people who have difficulty using a mouse or a keyboard.

- Keyboard overlays can make typing easier and more accurate for those who have motor control difficulties.

- Access to subtitled or sign language videos on the Internet for all deaf people

Accessibility guidelines

The problem arises when these tools need contents to be organized and structured in a specific way, such as screen readers, and this does not happen. In order to make sure Web developers do not perpetrate as many accessibility mistakes, as it is usual, several accessibility guidelines have been developed. The guidelines are textual descriptions of good practices, developers should follow in order to make their content accessible. Also they should check the contents of their Web pages, with theses guidelines, in order to verify that they are indeed accessible.

2.2.2 WAI and WCAG

The World Wide Web Consortium (W3C) is an international community where Member organizations and the public work together to develop Web standards. The objective is to ensure the long-term growth of the Web[8] by trying to enforce compatibility and agreement among industry members.

Web Accessibility Initiative

One area of efforts of the W3C is Accessibility. The Web Accessibility Initiative (WAI) is an effort to bring consensus to the Web and ensure the inclusion of everyone by making Web pages Accessible. For this it has developed strategies, guidelines, and resources to help make the Web accessible to people with disabilities. By concerning itself with Web pages accessibility, the WAI also includes accessibility for mobile devices in its scope. This is important since mobile devices have limited resources.

[8] http://www.w3.org/Consortium/

Web Content Accessibility Guidelines

There are several Web accessibility guidelines as different countries and entities have established different guidelines. An example of these guidelines are the American Section 508, the Italian Stanca Act, the German BITV1 and the Web Accessibility Initiative guidelines.

Known nowadays as the Web Content Accessibility Guidelines (WCAG) 1.0 and 2.0, these are some of the guidelines established by the WAI. This work only focuses on these guidelines specifically the WCAG 2.0. The reason for this is that these guidelines were developed through cooperation with the objective of establishing a single standard for Web accessibility with an international focus.

WCAG 1.0 and Priority Levels

WCAG 1.0 comprised a total of 14 guidelines and numerous checkpoints, organized into 3 priority levels, used to determine the accessibility of a Web page.

- Priority 1 or Level A Conformance with basic requirements for some groups to be able to use web contents.
- Priority 2 or Level AA Indicated better accessibility and removal of significant barriers to accessing content.
- Priority 3 or Level AAA Checkpoints provided improvements to web content accessibility.

This was a verification very HTML focused and as time passed and technologies improved checkpoints became less relevant and the development of WCAG 2.0 began.

WCAG 2.0 and Success Criterions

The WCAG 2.0 starts where the previous were left and introduces some significant changes. A shift in philosophy is noticeable since guidelines are now principle-centered rather than technique-centered. This change allows more flexible approach towards technological changes[9].

This version is composed of twelve guidelines, each one with some success criterion defined. These success criterions are passed or failed depending on the results of the

[9]http://webaim.org/standards/wcag/

different techniques they comprise. Failing one technique however, does not necessarily mean that the criterion will fail. Techniques, especially from different technologies, can make up for the failing of each other.

Techniques refer to, or are applicable to, different technologies such as HTML, CSS, Flash, PDF and others. In total there are eleven different types of techniques:

- General techniques
- HTML and XHTML techniques
- CSS techniques
- Client-side Scripting techniques
- Server-side Scripting techniques
- SMIL techniques
- Plain Text techniques
- ARIA techniques
- Flash techniques
- Silverlight techniques
- Pdf techniques

These guidelines were initially created for manual verification, but nowadays they have been coded as part of evaluation tools. These are known as accessibility automated evaluation tools and their purpose is to simplify the accessibility check process. There are several evaluation tools available nowadays and, although each follows specific guidelines and shows the results in a specific format, they all follow the same principle. As input, they receive a file or an active url, for the evaluation, they all run the coded version of the guidelines they employ and in the end, they present the results.

WCAG 2.0 CSS techniques

As nowadays the use of CSS is increasing developers are expected to separate HTML markup structure from their presentation in CSS files. As this happens, greater aesthetics in Webpages also appear, while at the same time, reducing the use of decorative images and, as consequence, download times[9]. This separation also provided flexibility, to control the presentation, but also style coherence, as multiple pages, shared the same style files.

However this increasing use of CSS did not necessarily result in an improvement of accessibility. One example of this, is when the developer defines styles that cannot be override and affect the styles defined by the user on the Browser[12]. Also functionalities provided by CSS properties misuse can lead to difficulties, an example of this is the use

of absolute units in the different properties making sizes unrealizable[10].

WCAG 2.0 establishes manual evaluation steps by technique in order to ensure CSS properties are correctly used. There are a total of twenty three CSS WCAG 2.0 techniques. Each represents a specific good practice that should be used by developers. For the evaluation stage, these techniques were individually studied and coded to produce a standardized return result. WCAG 2.0 CSS techniques include:

Techniques	**DOM Elements**
C6	Positioning content based on structural markup
C7	Using CSS to hide a portion of the link text
C8	Using CSS letter-spacing to control spacing within a word
C9	Using CSS to include decorative images
C12	Using percent for font sizes
C13	Using named font sizes
C14	Using `em` units for font sizes
C15	Using CSS to change the presentation of a user interface component when it receives focus
C17	Scaling form elements which contain text
C18	Using CSS margin and padding rules instead of spacer images for layout design
C19	Specifying alignment either to the left OR right in CSS
C20	Using relative measurements to set column widths so that lines can average 80 characters or less when the browser is resized
C21	Specifying line spacing in CSS
C22	Using CSS to control visual presentation of text
C23	Specifying text and background colors of secondary content such as banners, features and navigation in CSS while not specifying text and background colors of the main content
C24	Using `percentage` values in CSS for container sizes
C25	Specifying borders and layout in CSS to delineate areas of a Web page while not specifying text and text-background colors
C26	Providing options within the content to switch to a layout that does not require the user to scroll horizontally to read a line of text
C27	Making the DOM order match the visual order
C28	Specifying the size of text containers using `em` units
C29	Using a style switcher to provide a conforming alternate version

[10]http://www.w3.org/TR/WCAG20TECHS/C14.html

C30	Using CSS to replace text with images of text and providing user interface controls to switch

Table 2.1: Techniques summary description

As we will see ahead in this document, some of these CSS techniques have a highly visual implementation and because of this we decided not to implement them.

2.3 Evaluation Tools

In order to simplify the accessibility verification process, guidelines and success criterions have been coded as part of accessibility automated evaluation tools. These tools are given an URL as input to be verified according to the guidelines, and then produce an output with the result of this evaluation.

Although the process is always the same, different tools, perform evaluations according to different guidelines and thus produce different outputs. Together with the presentation of the final results to the user, this is what distinguishes these tools from each other.

In this section we introduce several tools developed by different identities and their characteristics.

2.3.1 AChecker

AChecker [13][14] Is a well known online tool, that evaluates HTML pages conformance with the BITV 1.0, Section 508, the Stanca Act and the WCAG 1.0 and WCAG 2.0 distributed in levels A, AA and AAA.

This tool is a semi-automated evaluator as it cannot verify all problems. To solve this errors are divided into three groups as can be seen in the image[11].

Problems in the Likely Problems section include the problems aChecker can not be sure and require the developer to make a decision.

Errors are presented by Success Criteria, identifying what is wrong and how the problems can be solved.

Although the tool has an option for showing CSS rules, CSS evaluation is provided by an external tool and does not present results according to guidelines, nor success criteria,

[11] http://achecker.ca/checker/index.php

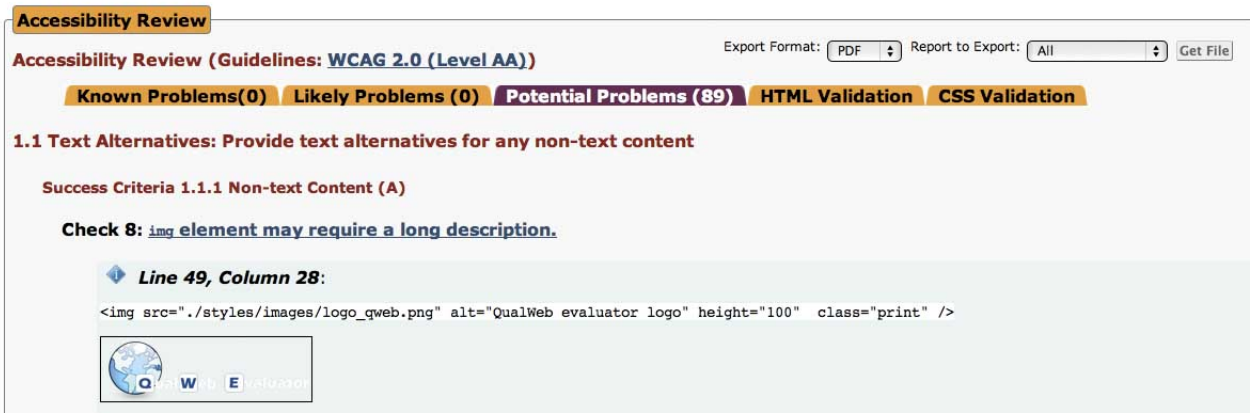

Figure 2.14: aChecker problems sections

nor techniques.

2.3.2 aDesigner

aDesigner [15] was developed by IBM and is available for download for WindowsXP or Vista. It checks Webpages according to the Section 508, the WCAG and the JIS (Japan Industrial Standard) guidelines.

It distinguished itself, from other tools, by doing something different. aDesigner simulates how low vision users see the Web pages by using ”low vision simulation” modes, and how blind users listen to and navigate through the pages by using ”blind visualization”. This simulation mode, allows developers to experience their Webpage as it would be perceived by other people.

An example of this simulation can be seen in the following figure[12].

Relating to CSS, aDesigner allows for a detection of potential usability issues in color contrast, font sizes, images and page links alternate text and makes recommendations.

[12]http://www.eclipse.org/actf/docs/users/aDesigner/docs/workspace.html

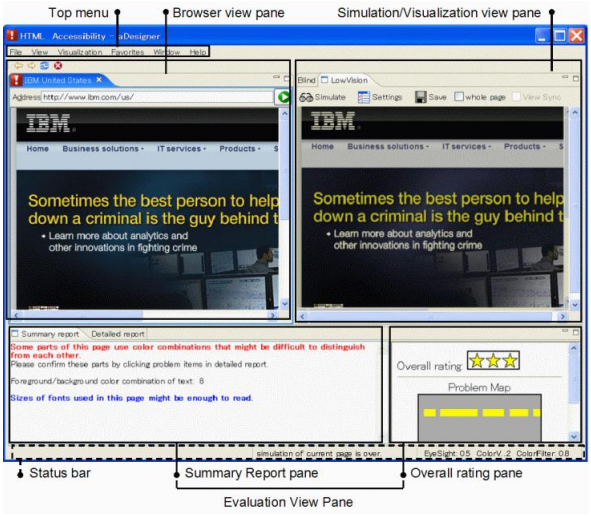

Figure 2.15: aDesigner simulation example

2.3.3 WAVE

WebAim, an organization that provides accessibility solutions such as articles, certification, consulting and this online evaluation tool known as WAVE[16].

For an evaluation result, WAVE presents a visual representation of the page being evaluated with yellow notifications. Notifications on the right side indicate accessibility problems. On the left side (Figure 2.16) a summary of the problems found is shown. This lateral report reviews the number of errors and possible errors and respective explanation.

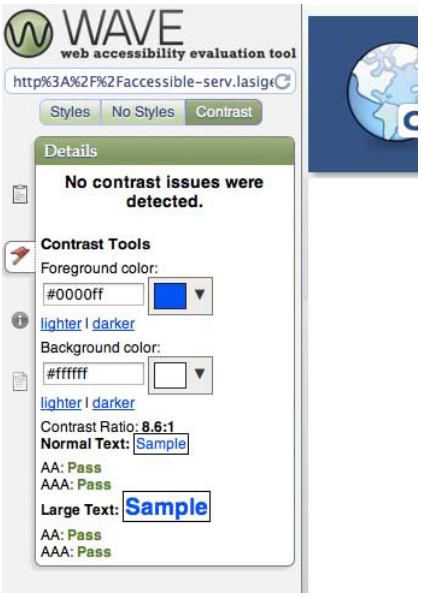

Figure 2.16: A CSS color contrast check by WAVE

The closest thing to CSS verification that WAVE performs is a check of the contrast ratio of the colors in the webpage. This can be seen in the previous figure[13].

2.3.4 HERA

Hera[17] is a Web-based and multi-lingual system that performs an evaluation with the WCAG 1.0 guidelines. First the user shown the results organized by guideline and inside each guideline, by success criterion.

These criterion results are colored accordingly to the result type. There are four result types, all can be seen in the two following figures [14]. Blue indicates checkpoints that need further manual verification, while green and red show final results for pass and fail respectively and grey signals criterions that were not applied.

Figure 2.17: Pass result

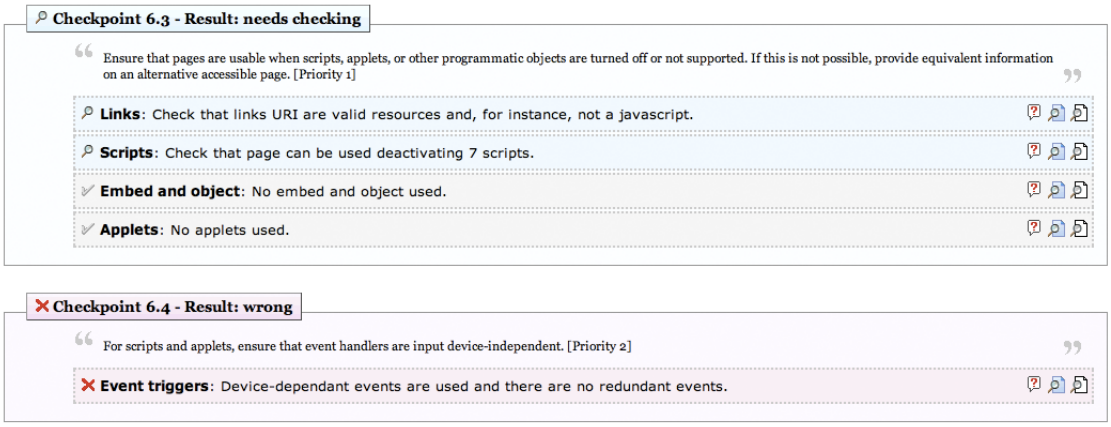

Figure 2.18: HERA Fail, Not applicable and Needs further verification

HERA also provides instructions on how to perform the tests, such as: ”There are 1 images without alternative texts. There are also 2 images that have an alt atrtibute. Check that the text included is a suitable replacement for the image.”

[13] http://wave.webaim.org
[14] http://www.sidar.org/hera/

This tool is very interesting since the display by guideline and then criteria, makes the structure of WCAG very easy to understand. Still regarding CSS evaluation, there is no specification of an evaluation being performed.

2.3.5 QualWeb

QualWeb[18] [19] is an accessibility evaluator tool for Rich Internet Applications (RIA). It uses the headless Browser PhantomJs from inside the NodeJs environment, in order to generate a post processing version of the Web page to be evaluated.

By evaluating this post processing of the Web page, it can deal with, and perform evaluation on, pages in their final state, after all scrips executions. This means evaluations are made over a Web page, as it would normally be viewed by the user, and not as it is in the source code. Doing this grants access to more elements, inserted by scripts, that would be inaccessible otherwise.

So far, QualWeb is a command line based evaluation tool, which implements 20 WCAG 2.0 HTML techniques. Relating to style sheets, QualWeb, also until now does not check CSS WCAG 2.0 techniques' compliance.

2.3.6 CSS Evaluation Tools

Online together with the previous tools mentioned there are also available a few solely CSS evaluation tools. These are mostly available online, some as browser extensions others as online tools that receive an url as input.

Currently although the use of CSS is increasing and thus increasing their relevance, few tools actually do heavy checking of CSS conformance. Also tools that are indeed available, are limited and mostly cover only basic color contrasts and color brightness checks.

The following are some examples of these tools.

- AccessColor [15] a free online tool which analyses the internal and external CSS of a web page to test the color contrast and color brightness between the text and background colors. An example of an evaluation can be seen in the following image[16]:

[15] http://www.accesskeys.org/tools/color-contrast.html
[16] http://www.accesskeys.org/tools/color-contrast.html

Figure 2.19: AccessColor example of repair

- CSS Analyser [17] That verifies color contrast and if sizes are specified in relative unit
- Color Contrast Analyser verifies the contrast ratio of two colors and classifies the pass or fail according to different level of guidelines. This is illustrated in the following figure[18].

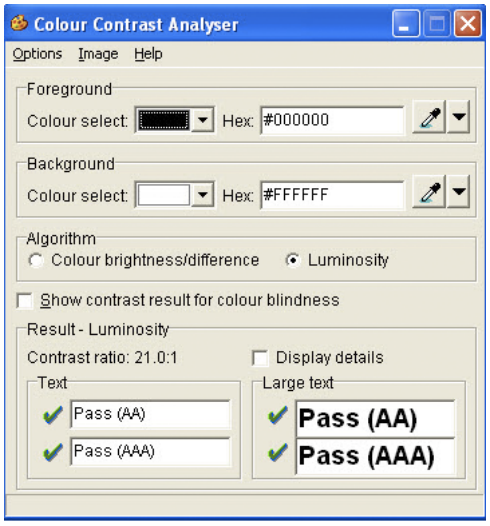

Figure 2.20: Colour Contrast Analyser Interface example

The problem is that none of these tools really do a complete style sheet checking according to specific guidelines.

2.4 Repair Tools

As a way to simplify automated accessibility evaluation, alongside Web evaluators, repair tools have started to appear. An automated repair tool has the purpose of repairing the ac-

[17] http://juicystudio.com/services/csstest.php

[18] http://www.visionaustralia.org/business-and-professionals/digital-access/resources/tools-to-download/colour-contrast-analyser-2-2-for-web-pages

cessibility problems found by an evaluation tool, that would otherwise need to be repaired manually.

The purpose of these tools is to repair, as many as possible, problems already identified by the evaluators, relieving developers of some work derived from the use of evaluation tools. Despite this these tools are not as common as evaluation tools probably because the repair process is not as smooth as the evaluation.

Techniques are textual descriptions of manual procedures, evaluation tools have to deal with ambiguous situations that sometimes can only be solved by human intervention. These situations are even more complicated to solve by a repair tool as few evaluation results can actually be repaired with certainty.

These repair tools follow specific guidelines, depending on the evaluation process, because in order to repair we need to first identify errors.

Some tools available online that we will mention here in this project are: HTML Tidy and The Social Accessibility Project by IBM.

2.4.1 HTML Tidy

The first, HTML Tidy, is a free tool capable of automatically fixing simple HTML problems, indent sloppy markup generated by editors and also identify some potential accessibility problems[17]. Developed by Dave Raggett of W3C (World Wide Web Consortium), Tidy is now a conjunct work of volunteers that are part of an open source community at Source Forge and users are encouraged to report bugs.

HTML Tidy allows input HTML in three forms: as a URL, as direct HTML code written or pasted into the text area, or as an uploaded file. Also the tool allows the user to select several options as setting for the repairing process. [19].

These options are of three types(Figure 2.21): HTML / XHTML, pretty print, and encoding. Options in the HTML / XHTML allow minor repair at the HTML level as well as the XHTML level, and some of them are:

- Break before BR - To output a line break before each
element.
- Drop empty paras - Removes empty p tags
- Fix Uri - checks attribute values that carry URIs for illegal characters
- Fix backslash - replaces backslash characters in URLs by forward slashes '/ '.

After uploading the HTML code and selecting the "Tidy" option, the repairs are done. The tool afterwards allows the user to either view or download the repaired page, presents

[19]http://infohound.net/tidy

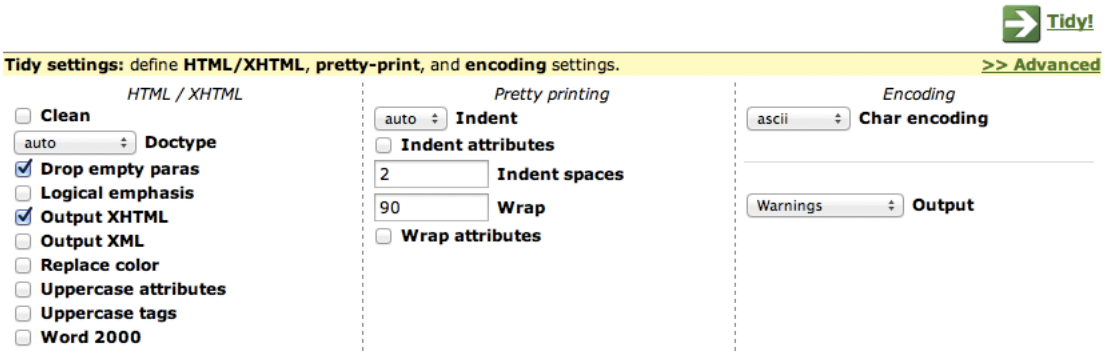

Figure 2.21: Tidy options

some summary information related to the repairing done; presents the repaired Web page itself and lists in yellow the warnings, all identified by a reference number.

Greater detail can be found at http://tidy.sourceforge.net/docs/quickref.html

2.4.2 The Social Accessibility Project

Another project relating to Web page repairing is the Social Accessibility Project by IBM Research Team at Tokyo[18], its main idea resides in the principle of communication between the user and the developer. Whenever a user has a difficulty on a webpage, he has the possibility of communicating it to the developer, with the expectation that the developer will correct the situation. The problem resides in the fact that most times this communication presents results very slowly and burdens developers.

The main purpose of this project is to help these users with difficulties while browsing, without going through Web developers. This Social Accessibility Project, presents a new model, in which the difficulties can be communicated and solved by a community of members. Who, on a volunteer basis, can discuss, create and publish the necessary metadata to solve the exposed problem. This way not only developers, but also users and community members, are able to improve accessibility by collaboratively authoring the accessibility metadata. The cycle of information can be completely understood in the figure 2.22, obtained from the IBM project webpage [20]

In this community there are users and supporters. Users are provided, after registering, with a client-side code, a plug-in. This plug-in will ensure the user's browser is able to open a dialog box for inputting the necessary text to describe the problem and to submit it. Whenever the User presses the send button in the plug-in installed, three things are sent to the community: the text written by the user, a snapshot of the screen with all

[20]http://www.research.ibm.com/trl/projects/acc_tech/index_e.htm

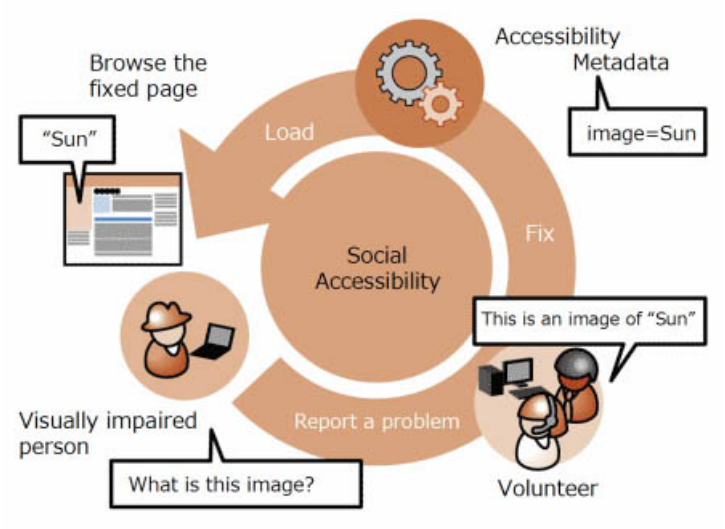

Figure 2.22: The Social Accessibility Project

inputs blurred, the reading position in the browser window and the XPath notation for that reading position.

Supporters, and authors of metadata, are provided with a browser extension sidebar. This extension makes notifications whenever someone sends a request for clarification. After pressing the ”start fixing it” button, the page in question is loaded with specific coloring to point out some obvious problems, to signal already corrected mistakes as well as headings.

Changes made by supporters are kept in a specific server to which the user is directed every time he tries to access the Web page repaired.

2.5 Tool Summary

The tools presented in section 2.1 are an example of the evaluation tools available online. We can understand so much from observing how they work, what guidelines they implement, how they implement them, how they present their results and how they deal with the same limitations that we faced and will face.

With this work we intend to expand QualWeb evaluator’s functionalities. As described, QualWeb already has great potential, however still lacks WCAG 2.0 CSS techniques implementation.

As we have seen in previous sections, despite growing use of style sheets, current evaluation tools and repair tools, still lack a true CSS WCAG 2.0 implementation. Enhancing QualWeb will make it even more competitive.

Regarding previous examples of tools, we can see that each of them present different approaches to web evaluation and repair.

- aChecker is a well known tool but makes CSS evaluations through a different entity;
- aDesigner is a great and different concept but does not actually focus on CSS techniques. Rather it focuses on visualization of Web pages as a whole by simulating environments;
- WAVE and other CSS exclusive tools have only basic color contrast evaluation and do not verify specific CSS techniques;
- HERA has a very interesting and easy to understand interface, but does not specifies if it checks CSS accessibility;

Regarding the mentioned repair tools, they are good examples, with interesting functionalities. But they fall short considering what we want to do.

- While HTML Tidy is a valuable tool for simple HTML / XHTML and presentation repairs, its main concern is simple html structures and does not follow any specific accessibility guidelines. Considering our project Tidy is viewed as a good concept with a simple presentation of the results and the downloading of repaired files.
- The social accessibility project is a very interesting concept, that shows how the power of a community can the brought together to help improve the Web. However, unlike what we propose to implement, the social accessibility project does not follow specific guidelines, but rather is based on an asking for help and then solving the problem approach, not to mention that it needs constant availability of volunteers.

These two projects differ considerably from what we intend to do, despite the final concept being so similar. We propose that our repair tool presents a set of recommended repairs, for as many error as it can. Also we propose that our repair tool, like the QualWeb evaluator tool, will follow the established WCAG 2.0 guidelines.

Despite all the differences, one thing that seems common to all automated tools is the acceptance that it is not possible to identify or repair every single type of mistake. aChecker asks the developer for confirmations; Tidy presents a list of warning which signal errors that could not be repaired and the social accessibility project requires nothing but human interaction to repair the web. This means that to some extent, we need to acknowledge too, that it will not be possible to make our repair tool 100% automatic.

Chapter 3

Architecture

By the time we started this work, QualWeb was an automated accessibility evaluation tool for RIA, which implemented WCAG 2.0 HTML techniques. It was a command line application, but has since then, grown into an online tool, through the development of a user interface. Despite this, it can still be used as an command line based tool in order to perform large scale evaluations.

With this work we have expanded QualWeb, so that it is able to perform style sheet evaluations and repair suggestions, as well as present these results. This required adding some new components to the initial QualWeb's architecture, which can be seen in greater detail in figure 3.1.

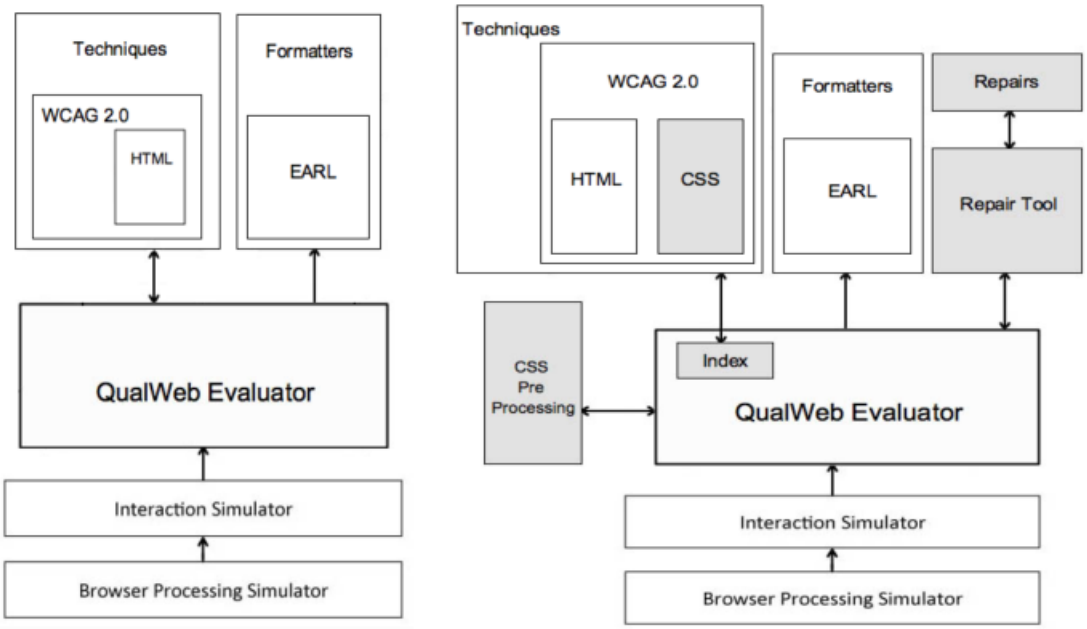

Figure 3.1: QualWeb's architecture before and after this work

3.1 Modules

In order to implement our two main objectives, established in the beginning: evaluating and repairing the Web, we had to develop all the highlighted modules on the right side of figure 3.1. These four modules are:

- The WCAG 2.0 CSS techniques modules;
- CSS Pre-Processing module;
- Index module;
- The repair module.

Although they are individually presented in the previous list, the first three are derived from our first objective, which was to evaluate the Web. Despite this each one has a specific role to play in the evaluation process and in this work.

3.1.1 The WCAG 2.0 CSS Techniques

Initially developed as textual descriptions by the WAI, they describe procedures to be followed in order to check specific elements. For example, technique H37 states that image elements should have alt attributes. A validator (human or automated) is expected to run through all the HTML elements, find each image element and then check if it has an alt attribute.

CSS techniques' textual descriptions, were individually coded transposed and added to the evaluation process. These can be identified in the architecture, figure 3.1, in the CSS techniques section. As we can see, QualWeb is structured in a way that allows for easy implementation of other techniques and guidelines.

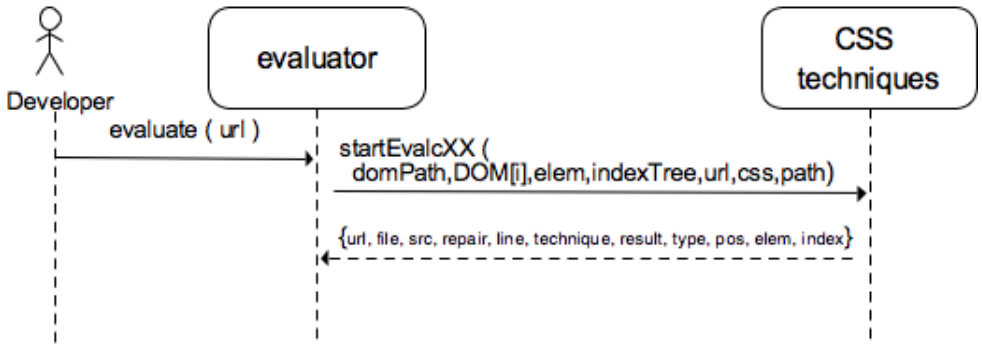

Figure 3.2: Input & output requirements for CSS techniques

Figure 3.2 illustrates and specifies the input and output specifications. Data flowing between the core of the evaluation tool and each of the CSS techniques must include

all these attributes. This allows for coherence between CSS technique implementations. Also, returning the same error format is important for later usage by the repair tool and for the interface.

3.1.2 CSS Pre-Processing

When we started implementing the CSS techniques, we realized we would need to obtain the CSS rules of each element to be evaluated. As the PhantomJs tool supposedly functions as a headless Browser, we expected these CSS to be inside the DOM already. However, soon we realized this was not the case, PhantomJs does indeed create a DOM, but only keeps the inline CSS. This makes somewhat sense since they are the only CSS that are part of the HTML structure as they come as elements' style attributes.

This is a paragraph.

In order to verify the compliance of the code, we needed to have all the CSS information and since some CSS were missing, we had to develop a module responsible for gathering and formatting them. This module was called the CSS pre-processing and is executed before the evaluation process itself. Its responsibility is to ensure that, during the evaluation process, QualWeb can access every CSS specified for the current element of the iteration including: <link>and <style>CSS.

Because these two types of CSS are obtained in string format, which is very unpractical to use, this module will also be responsible for organizing them into a structure. This conversion ensures that the CSS will be prepared for later processing during the evaluation.

3.1.3 Index - Integration of Evaluations

Another subtask that came up during the evaluation stage, was the integration of CSS evaluation and the HTML evaluation. In the beginning of this work, each HTML technique individually run through all the DOM. For example:

Technique H37 states that img tags must have alt attribute. When implemented, the technique finds img tags in the HTML document, and checks for the existence of an not null alt attribute. This means that the implementation has to parse all the HTML elements; find every img tag and then verify its attributes.

This causes techniques to repeat code and to spend unnecessary processing time. However, since most techniques are local, to an element, and require only parent and sibling information we decided to alter this paradigm. From now on the DOM would be

run trough in a separate file, from where techniques would latter be triggered from. For this to work, while CSS techniques were already implemented with the new paradigm in mind, HTML techniques had to be adapted(Figure 3.3).

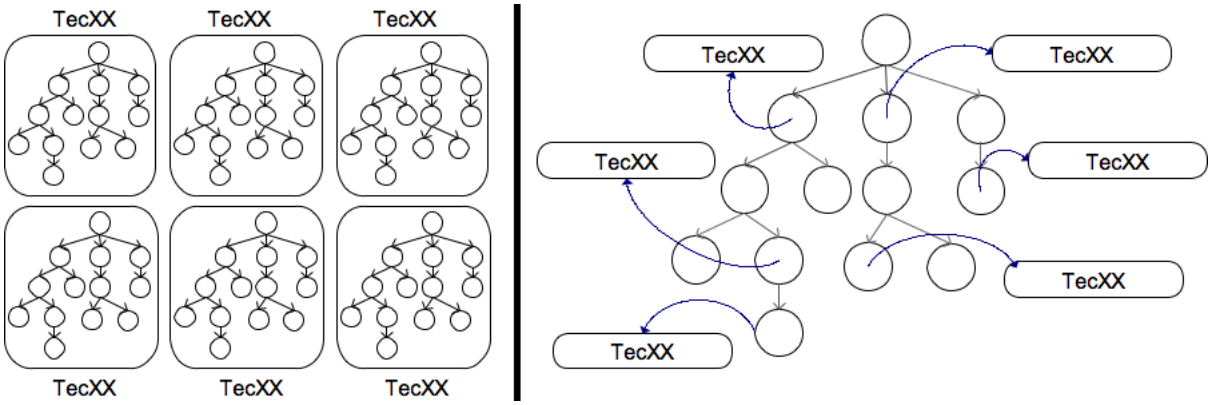

Figure 3.3: DOM execution before this work and how we are going to changed it

This work changed how QualWeb approaches techniques. Using the same example as before, now, while running through the HTML elements, if it finds an img tag, checks whether there is an alt attribute, or not, and if its value is different from null(Figure 3.4).

To the separate file that iterated through the DOM and from where techniques are triggered, we gave the name of index file (Figure 3.1). This optimized QualWeb execution since each technique does not need to parse the entire HTML document anymore.

Another important thing is that this change in perspective, allowed QualWeb to be more flexible. Until here, QualWeb could only return to the user the evaluation results ordered by technique. Now, results can be kept, and returned to the user, in the same order as they are found in the DOM. This brings two major advantages:

1. By presenting results as they occur in the Web page, the developer is allowed a better workflow with continuity along the Web page;
2. Results are now flexible enough to be ordered depending on what is chosen in the Interface.

3.1.4 The Repair

In figure 3.1 we can also view the new repair module. It is divided onto two parts, the repair tool and the repairs component. The repair tool is used for keeping every function it is responsible for, while the repairs component is only responsible for building the repairs.

During the evaluation process, the DOM's elements are iterated and a determined number of techniques is triggered in this process. Each time, an output is produced and

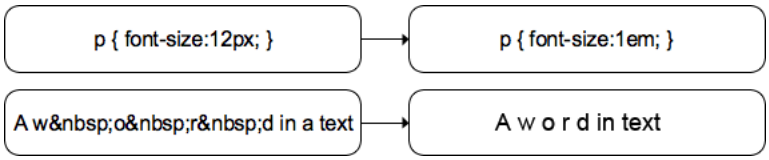

Figure 3.4: Two examples of algorithms applied in the repair process

kept as an entry of the final results structure. Each entry of the output result will have a set of attributes, which the repair module will be responsible for using, in order to establish what repair to use.

The objective is to create a string with the possible steps the user can follow in order to repair the problem found during the evaluation process. However, for the building of these textual steps, sometimes unit conversions and character removing had to be done. Figure 3.4 illustrates two simple examples.

This module requires an evaluation to run beforehand, as it must receive the results structure as input. Also since repair steps are to be interpreted by a developer,experienced or not, repair suggestions must be written in simple language.

3.2 Module Interactions

Figure 3.5 depicts a overall view of the implementation, connection and interaction between different components. It shows the four components. The execution starts with the run.js file which does part of the CSS pre-processing procedure, by obtaining internal and external CSS (CSS in <style>and <link>). The CSS formatting is done in the readCss.js file.

Afterwards starts the evaluation itself, through the evaluator.Js, which is responsible for running the DOM iteration and calling the necessary techniques (both HTML and CSS). All techniques interact with the evaluatorCssLookUp.js file, but different ways are specified in the detailed annexed collaboration diagram.

On the right, we can see the execution of the repair module. Separation between repairTool.js and the script that generated the repairs themselves, was envisioned in a way that allows other functionalities to be added to the repair tool, while still keeping modularity.

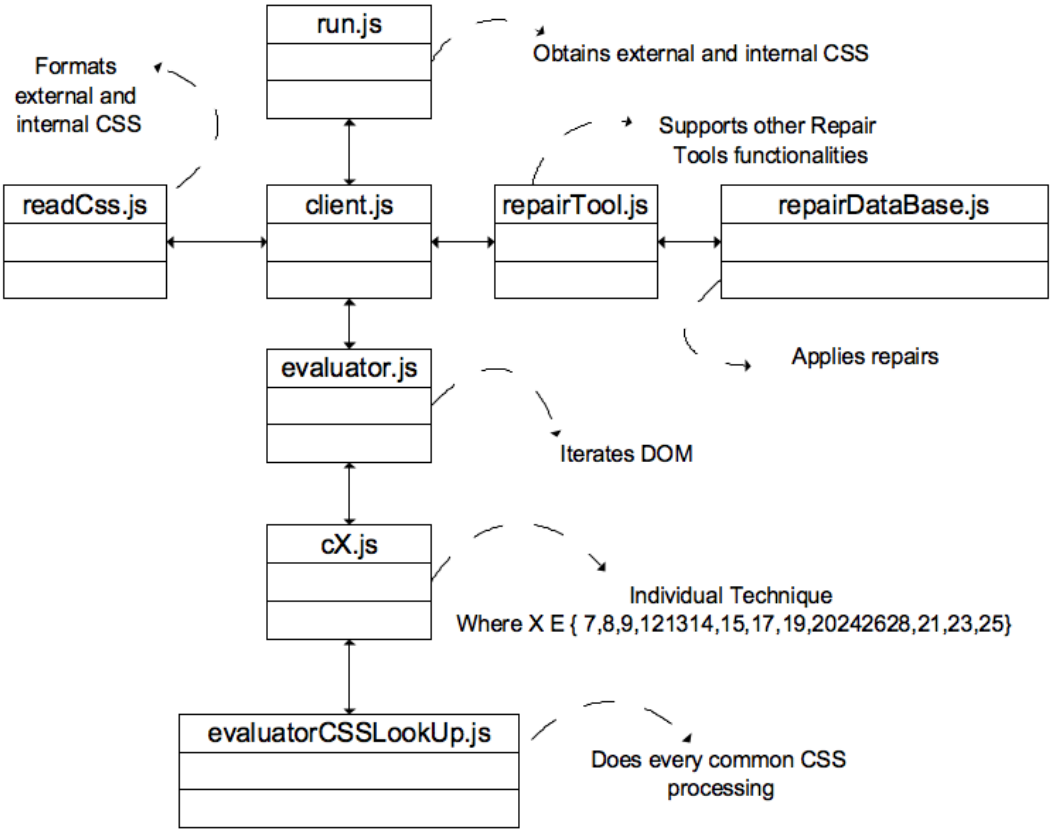

Figure 3.5: Implementation Diagram

Diagrams with greater implementation details were annexed to this document. The first shows all the different elements implemented, their functions and attributes, the second is a collaboration diagram which illustrates how all these different elements interact between each other in a normal execution environment.

Chapter 4

Gathering the CSS

Before advancing into the Evaluation stage of this project, we needed to obtain all the styles to be applied to the Web page. As we have seen before (Chapter 2) there are three different types of CSS each introduced, into the Web page, in different ways and rendered with different priorities.

This chapter details the different ways we obtained the CSS, why some of these CSS needed to be formatted into a specific structure while others did not.

4.1 CSS pre-processing

This procedure refers to the process of gathering <link>and <style>CSS. <link>are introduced into the Web page though references while <style>are inside a style HTML tag. Both types are retrieved in plain text, which means they require structuring for later matching with the HTML elements.

This section also explains how and why we obtained and formatted these CSS and figure 4.1 depicts this process as well as how these components must interact with each other.

4.1.1 <link>CSS or External .CSS Files

The first type of CSS to be gathered are the <link>CSS. These CSS generally appear on the head section of the Web page. Retrieving these CSS was one of the greatest challenges during this work and the reason this task was prolonged so much.

<link href="../indexcA.css" rel="stylesheet" id="css" type="text/css" >

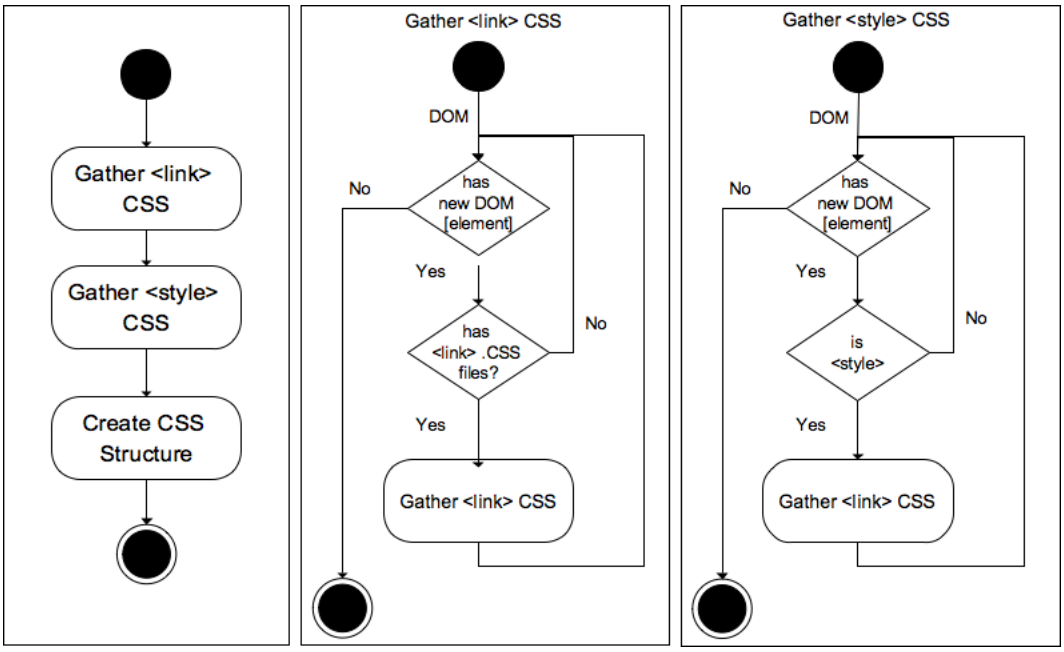

Figure 4.1: Pre processing of the CSS

Our purpose was to retrieve each .css file and copy its content into a local file. Since the project was being developed inside the NodeJS environment and these .css files had to be retrieved as external files we needed to use a node function that would retrieve a file given it's URL. Since we were not able find one specific command, we used the node function exec, that allows a curl command to be executed. This curl command fetched the document based on its URL:

$$child = exec(\ curl\ -\ L\ +\ url,\ function\ (error,\ stdout,\ stderr))$$

However soon we realized that doing this was not enough. Retrieving the CSS using the exec function implied creating a child processes, and since NodeJs executes everything in an asynchronous way; the parent process would continue to execute without having the final file written.

To make sure these exec calls were done, and the CSS were retrieved, before the evaluation itself (the parent process), we decided to run the evaluation in the last exec call. This way we created a separate child process for the evaluation which would have access to that final .txt file where the file's content was kept.

This provided a fair solution, however in the end turned out to be ineffective on a global sense of the QualWeb project execution. All these extra processes, one per file, made the execution really heavy.

Currently a better solution has been found and QualWeb now retrieved these contents through the use of a request module, installed to NodeJS. This request module allows Node js to perform HTTP Requests.

request(cssUrls[index], function (error, response, body)

In the end no exec function needed to be used.

4.1.2 <style>CSS or Internal CSS

These CSS come on the top of the Webpage inside a <style>tag. They are easily obtained since it only requires the algorithm to find a style tag to retrieve its content and concatenate it to the previous CSS. These, although being easy to obtain through the use of the DOM structure, also feature in this section because they are put together with the previous CSS to be formatted and then processed.

4.1.3 Structuring the CSS

Once all the CSS, except the inline CSS, were joined in a single variable, they were organized into a specific structure that allows an easier access to the contents during the evaluation process. This structure makes sure every CSS rule and comment line or comment block, is kept as an entry with the following format:

A comment block in the structure:

[{ index: 0,
file: 0,
css: ‘’,
comment: true,
lineNumber: 3,
initialCss: ‘\t \t \t /∗\n \t \t \t One long long \n \t \t \t long \n \t \t \t long comment \n \t \t \t with some { and some } inside \n \t \t \t for testing... # \n \t \t \t ∗/ \n ’}

A simple CSS block in the structure:

{ index: 15,
file: 0,

css: ‘p{letter-spacing:1;font-size: 20%;} ’,
comment: false,
lineNumber: 24,
initialCss: ‘\t \t \tp {letter-spacing: 1pt; font-size: 20 %; } \n’}]

A slightly more intricate CSS block in the structure (this block has CSS rules in different lines):

{ index: 70,
file: ‘http://localhost:8093/tests/c0/indexcB.css’,
css: ‘#tableB{width:100%;height:75px; }’,
comment: false,
lineNumber: 6,
initialCss: ‘\n \t #tableB { \n \t\t width:100%; \n \t \t height:75px; \n \t } \n’},

Overall this structure has a new entry for every css rule, line or block of comments found, and for each entry keeps the following information:

- An index, which is the position of the excerpt of code in the structure;
- An attribute named file, which keeps the file in which the excerpt of code was found;
- The comment attribute, that allows, when looking for CSS rules, for a more immediate exclusion of comment entries;
- The lineNumber which indicates the line where the excerpt of code began. In the above examples, looking at the information kept in the initialCss attribute we can see how a CSS rule or a comment block can also include several lines;
- Finally the attributes initialCss and css. The initialCss attribute keeps a detailed and exact version of the code, whilst the css variable keeps a clean version. It is important for us to have both, as the first ensures that once the css is repaired, replaced into the document or shown to the user, it will stay exactly as it was before; and as the second one will be easier to process during the evaluation algorithm, that will be detailed ahead.

During the evaluation process a correct matching between HTML elements and the corresponding CSS rules is made, this process is detailed ahead in the Evaluating the Web chapter.

4.1.4 Postponing CSS Full Processing

In this chapter we detailed how we dealt with the different types of CSS. The first two types require retrieving and formatting, since they do not show in the DOM, while the inline CSS does not.

Along they way we though of processing all these CSS, in the beginning. This way we would place them in the DOM, with the inline CSS. Nevertheless this idea did not move forward. The WCAG 2.0 CSS techniques describe evaluation procedures for specific elements. By analyzing these techniques (Chapter 5) we will see that some HTML elements do not trigger any CSS technique. If this approach went forward we would be matching CSS with elements that possibly would not be evaluated, thus spending processing time unnecessarily.

4.2 Inline CSS

Excluding the CSS marked as important, these CSS are the ones with highest priority and because of this, can be used to override other CSS. They are the easiest to obtain, from the DOM structure. They already come attached to the element they refer to and do not need to be included in the processing stage. The following description corresponds to a DOM element and its CSS can be accessed directly as: [”attributes”][”style”].

{ name = ”p”
attributes = { id=”first paragraph”; style=”line-heigh:1.5em;” }
children = {Object; Object; Object}}

4.3 Scripted CSS

Scripted CSS are CSS dynamically loaded with the execution of scripts, for example:

function colorElementRed(id) {
var el = document.getElementById(id);
el.style.color = ”red”; }

These scripts are run only when and after the rendering of the Web page by the Browser, and most evaluators will never have access to them. This does not happen in QualWeb, since it uses phantom.js to process the Webpage as a Browser would.

Chapter 5

Evaluating the Web

This chapter intends to explain the entire process of evaluation, from the interpretation of the CSS techniques, until the evaluation of the HTML document.

First we had to understand the WCAG concepts, types of results and techniques. As specified in the State of the Art chapter, there are several CSS techniques and interpreting them was the first step of the implementation. From these interpretations, we then derived what tags would trigger them.

Next step was the implementation stage. The objective was to transpose as much as possible the textual descriptions in the WCAG 2.0 Web page. It is important to mention, that due to their highly visual component and complexity, some techniques were not implemented.

5.1 Interpreting WCAG 2.0 CSS techniques

These CSS techniques were developed as visual verifications. Many include statements as ”Using a mouse, hover over the element.”, ”Using a keyboard, tab to the element.” or even ”Remove the style information from the document or turn off use of style sheets in the user agent.” and ”Verify that the resulting page is a conforming alternate version for the original page”.

Because of this, these description into coding checking, became a challenge that sometimes we decided not to overcame. Techniques C6,C18,C22,C27,C29 and C30 were examples of these situations.

Also some of the techniques, due to their similarity, ended up being merged together. This way, for a specific element, similar techniques could be checked. This happened to techniques C12 C13 and C14 and C20, C24, C26 and C28. Merging these techniques

avoided strange sequential evaluation results(Example can be seen ahead).

5.1.1 Tecniques

C6

Techniques C6 has the objective of demonstrating how visual appearance may be enhanced via style sheets while maintaining a meaningful presentation of content. The objective is to make sure that whenever a user decides to disable styles on a browser, the Web page still makes sense since it has a understandable structure that makes sense.

Figure 5.1: Styled List (Left) Same list without CSS(Right)

Procedures to test the compliance of the Web page, according to this technique include:

1. Remove the style information from the document or turn off use of style sheets in the user agent.
2. Check that the structural relations and the meaning of the content are preserved.

Because of these steps, this was the first of the techniques that we chose not to implement.

C7

This technique's objective is to supplement the text inside anchors with additional text that describes the unique function of the link. This additional text needs to be styled so that it is not rendered on the screen. When information is needed to interpret the displayed link text, this technique provides a complete description of the link function while allowing the less complete text to be displayed.

An implementation needs to verify three things for every anchor:

- a child span tag exists;
- a description for the link is inside that span child and

- a CSS rule exactly as ”a span height: 1px; width: 1px; position: absolute; overflow: hidden; top: -10px; ”

C8

The objective of this technique is to ensure special space characters such as the `& nbsp` character, while being used to provide spacing between letters, are replaced by the use of the letter-spacing property. This is recommended since the blank characters can change the meaning and pronunciation of the word. For this technique we want to identify these space characters in the middle of words.

C9

Technique C9 has de objective of providing a mechanism to add decorative images without requiring additional markup. This way the user by removing styles can make these images disappear. WCAG identifies a decorative image as an image used for visual formatting of Web content and assumes a decorative image is identified in the HTML image tag with an empty alt attribute.

Some other ways to identify these images could be used, for example: by heights or widths, since decorative images can be used to create spaces from the top or from the left, and size of the image since these images are generally the same color of the background their size is supposedly smaller.

C12, C13 and C14

These are one example of the techniques that were merged together:

- C12 recommends ”Using `percent` for font sizes”;
- C13 recommends ”Using named font sizes” and
- C14 recommends ”Using `em` units for font sizes”.

The reason for this merging is based on the fact, that when an element was verified, it was verified three different times. For example, when a DOM element was a paragraph, which had, lets say, font-size in `em` units, evaluation results would be: Technique C12 fail, Technique C13 fail, Technique C14 pass. Not to speak that in the repair execution, this paragraph’s CSS would either need repairs in the two first occurrences.

C15

The objective of this technique is to demonstrate that visual appearance may be enhanced using CSS, in order to provide visual feedback when an interactive element obtains focus or when the user hovers over it using a pointing device. By highlighting the element information can be provided to show the element is interactable.

Although this technique has several actions impossible to simulate using a automated tool such as: ”Using a mouse, hover over the element.” and ”Using a keyboard, tab to the element.”. In this case we were able to direct this into checking for: ”onMouseOver” and ”onfocus” inline properties and ”:hover” and ”:focus” in CSS selectors.

C17

Ensures text-based form controls resize when text size is changed in the user agent. This will allow users to enter text and read it in the input boxes, because the text is displayed at the size required by the user.

For this technique verification steps are:

1. Enter some text into text-based form controls that receive user entered text.
2. Increase the text size of the content by 200%.
3. Check that the text in text-based form controls has increased by 200%.

In this case also, we were able to transpose these steps into other types of verifications, according to the examples given in the WCAG 2.0 technique’s description Web page. What we verify is the usage of relative units in font-sizes, which allow Browsers to resize the text.

C18

The purpose of this technique is to verify the incorrect use of spacer images (usually 1x1 pixel, transparent GIFs) in tables or to indent a paragraph. This is incorrect since margins and paddings can be used on their own or in combination to control the layout this way. Images as we mentioned before would be impossible to remove and would affect the functioning of screen readers.

This technique was also not implemented since we could not be sure if an image was a spacer image.

C19

This technique functions more or less as the technique C8. The purpose is to specifying alignment either to the left or right in CSS, by avoiding
texttt& nbspcharacters. Blocks of text either left or right by setting the CSS text-align property.

Here we check if multiple
texttt& nbspcharacters are found consecutively in the beginning of a line, a situation which could be solved though the use of an align right or left property. Also in this technique we check if the text is justified or centered as the technique also says this can cause accessibility difficulties.

C20, C24, C26 and C28.

As techniques C12, C13 and C14, these four rules were also merged together since they all refer to containers sizes:

- C20 recommends ”Using relative measurements to set column widths so that lines can average 80 characters or less when the browser is resized”
  It has the purpose of ensuring CSS is used in a way that allows users to view content in such a way that line length can average 80 characters or less. Allowing users with certain reading or vision disabilities that have trouble keeping their place when reading long lines of text to view and interact with the content more efficiently.

- C24 recommends ”Using `percentage` values in CSS for container sizes”.
  It specifies that developers must define widths of text containers using percent values, in order to enable users to increase the size of text, without having to scroll horizontally to read that text.

- C26 recommends ”Providing options within the content to switch to a layout that does not require the user to scroll horizontally to read a line of text”.
  It states developers should use containers that do not require the user to scroll horizontally to read a line of text. This can be verified also through the checking of percent values.

- C28 recommends ”Specifying the size of text containers using `em` units”.
  It states developers must create containers that by specifying width and/or height in `em` units. This will allow user agents that support text resizing, to resize the text containers. Also reduces the probability of text cropping when text size has been increased so that it falls outside the container boundaries.

As an implementation, we look in the CSS for the container, for a with property. If it exists, it must have relative units: `percentage` or `em`. Each line of text must also have less than 80 characters.

C21

This technique makes sure developers provide line spacing between 1.5 to 2. This, allows users with cognitive difficulties in tracking a single spaced line in a block of text, to, once they have finished the previous one, start reading a new line more easily.

In this implementation we look for line spacing specification in paragraphs and if it has the correct values.

C22

Technique C22 objective is to demonstrate how CSS can be used to control the visual presentation of text. This will allow users to modify the visual characteristics of the text through style sheets. The objective is to show developers that they do not need to use images of text in order to style it. For this, they can use a set of properties, such as font-family, font-style, color, line-height, text-transform, etc.

This is one of the techniques that were not implemented. Detecting if an image was used in order to provide text with enhanced styling, would require specific tools for image content analysis.

C23

The objective of this technique is to inform developers, that although they can specify text and background colors of secondary content, such as banners, features and navigation in CSS, they should not specify text and background colors of the main content.

This allows users to select specific color combinations for the text and background of the main content, according to their needs, while retaining visual clues to the groupings and organization of the web page.

Here is important to check if there is any color specification, in the main content area.

C25

Simillar to the previous technique the purpose here is specify borders and layout using CSS and leave text and background colors to render according to the user's browser and/or

operating system settings.

This will allow users to view the text in the colors they require while maintaining other aspects of the layout and page design such as columns of text, borders around sections or vertical lines between a menu and main content area.

Similar to technique C23 the objective here is to check if there is any color specification, in the main content area.

C27

Also not implemented, technique C27 verifies if DOM order matches the visual order of the Web page. The order of content in the source code can be changed by the developer.

This change may cause confusion for assistive technology users, when accessing the content directly from the source code (such as with a screen reader), or by interacting with the content with a keyboard.

For example: A user with low vision who uses a screen magnifier in combination with a screen reader may be confused when the reading order appears to skip around on the screen. A keyboard user may have trouble predicting where focus will go next when the source order does not match the visual order.

Steps to verify this technique are:

1. Visually examine the order of the content in the Web page as it is presented to the end user.
2. Examine the elements in the DOM using a tool that allows you to see the DOM.
3. Ensure that the the order of the content in the source code sections match the visual presentation of the content in the Web page

By reading these steps it is easy to understand why we chose not to implement this technique. Steps to be followed are mainly visual, and checking content presentation order would be possible but it would require us to estimate, by understanding CSS, where the browser would place each item.

C29

This technique refers to style switchers to provide a accessible alternate version of a Web page. This way whenever some aspect of the default presentation of a Web page is not accessible, it is still possible to meet that requirement by using an ”Alternate Version” of the page.

It would require us to be able to detect a style switcher method and to verify if the CSS implemented are accessible. Since this would consume lots of time we decided to also not implement this technique.

C30

The objective here was to use CSS to replace text with images of text, and provide controls to switch. This way, the capacity of CSS to replace structured HTML text with images of text is used and allows users to view content according to their preferences.

To use this technique, an author starts by creating an HTML structured page. The author then designs two or more stylesheets for that page. One to present the HTML text and the second, uses CSS features to replace some of the HTML text with images of text. After this, through the use of a style switcher, the author provides a way that allows the user to switch between both. This technique was also not implemented due to its complexity.

5.1.2 Triggering CSS techniques

From the textual descriptions of the techniques, we established a set of attributes that whenever found in the DOM structure, would trigger the execution of the implemented techniques.

Techniques	HTML Elements
c6	————————
c7	<a>
c8	<p><h..>
c9	<img>
c121314	<p><h..>
c15	<input><a>
c17	<input><button><label>
c18	————————
c19	<p>
c20242628	<body >
c21	<p>
c22	————————
c23	<body>
c25	<body>
c27	————————
c29	————————
c30	————————

Table 5.1: Techniques activation by tag

5.2 The Evaluation

This section is intended to describe in detail, the implementation process of the techniques. Including how the CSS are obtained for each element being evaluated.

5.2.1 Inputs

For the modularization of the code, each technique's code was kept in a separate .js file and as the elements of the DOM are run through, these separate files are triggered. To each of theses files a set of specific attributes is sent, so that the techniques algorithm is able to run and the report results created. These inputs are shown in table 5.2.

Argument	Description
path	The list of elements passed by. Used for determining the parents of the element and subsequently the matching CSS
list	The DOM containing the current element's information and its children
elemName	The name of the current element
k	An index also for identifying the parents of the current element
u	Url to be evaluated
css	The whole CSS information
pathG	Path to be used for importing external files

Table 5.2: Technique Inputs for the Evaluation

5.2.2 Process

These .js files, triggered during the iteration of the DOM structure, follow steps in order to evaluate the current element. These steps can vary depending on the technique but overall they follow this procedure:

1. The first step is to build a string containing the element and its attributes. This information will be used in QualWeb's interface for identifying the element where the triggering of the technique occurred. This string will include the element and a concatenation of all its attributes.

2. Next we need to gather all information about the element in order to identify the appropriate CSS rule to be applied to that element. This includes attributes class or id and the elements' parents. Elements parents are other HTML elements, inside which the current element is located, according to the HTML hierarchy system.

3. This step may not always be needed, depending on the technique. It includes, checking inline CSS and, if it does not exist or if it does not have the property we are looking for, other CSS rules for the current element.

4. Depending if the previous step is triggered or not, the next thing to do is the evaluation of the CSS found. This includes checking for the existence of the property we are looking for or checking whether it has a specific unit.

5. The last thing to be done is the formatting of the result, depending on the previous two steps.

Since each .js file is triggered by a single element, its execution is repeated for every element of the same type. For example technique c15 is executed once for every anchor or input element found in HTML code. This means that for every execution of the technique an error report entry is generated. All these entries are then kept together in a final return structure that will be returned to the repair tool and to the user interface.

5.2.3 Retrieving element's information

In order to identify which CSS are associated with the current element, we need to have as much information about it, as it is possible. Typically an HTML element is identified by the id or the class attribute, but this is not always true. To accurately pinpoint which CSS applies to an element we also need to know its parents. For example, an anchor element without id or class attributes, is not just an anchor, it is an anchor perhaps located inside a paragraph, which is inside a div, that may or may not have a class or id attribute. And this can be its only identification, for example: anchors inside the div with id #maincontent

The DOM is a tree structure, where nodes have no reference to parent nodes, and the algorithm used to go through elements is a depth-first algorithm. This is a problem, since it means that from each element, we cannot obtain information about who the parent is.

The solution we found, was to use a list to store every element run through. The difference is that, each entry keeps the attributes of the object and its children as well as a new attribute. This new attribute is used to keep a number representative of the element's depth. Figure 5.2 shows the structure in 5.3 this list.

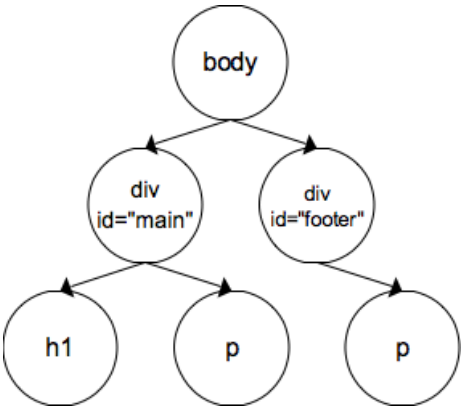

Figure 5.2: DOM tree of elements

Using this variable to get the parents is simple to understand. For example, if the HTML tag has a 1, the head and the body tags both have a 2. This way we know that HTML is a parent of head and body and that both are brothers. So in order to get parents, we would go backwards on the variable, and save all elements with lower depth. For example for element p in figure 5.2 we would have parents: body and main.

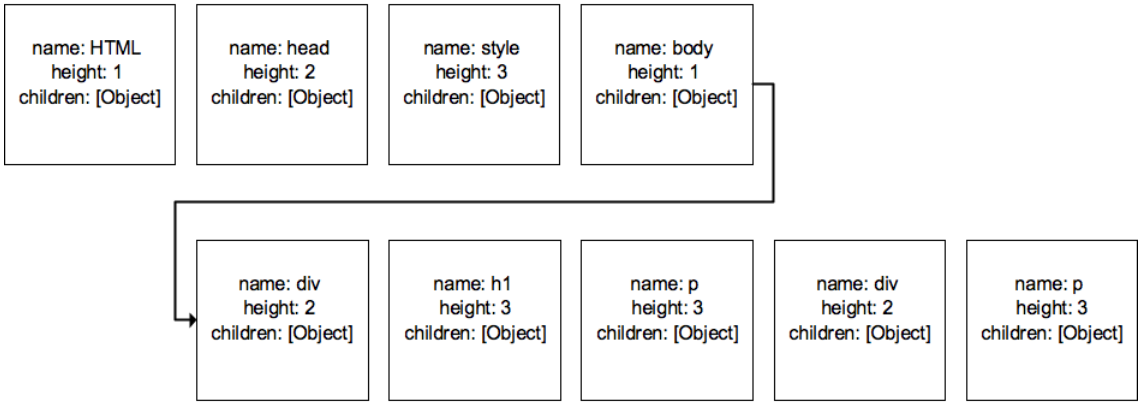

Figure 5.3: Variable with list of elements

Thanks to this approach, not only can we, given an element, search for its siblings and parents and choose how far up we want to go in the family, it also allows us to look for a specific property in different elements. For example, font-sizes for a paragraph element can be defined in any of its parents. So if you have a paragraph inside a div, which is inside the body element, you need to search for that property in every single one of those elements.

Again using figures 5.2 and 5.3: Parents list for p element is #main body and imagining we are looking for the font-size property for paragraphs, the two rules 1 and 2 are not the same thing:

1. div p{ font-size:2em;}
2. #main p{ font-size:4em;}
3. body{ font-size:20px;}

Rule 2 is the best CSS to be chosen for p element, while rule 1 should only be chosen if font-size property is not found in number 2 rule. This also means that rule 3 must also be considered for this element, just in case no font-size property is found on any CSS for paragraphs, nor on the CSS for paragraphs inside divs with id: main, and not even on CSS for paragraphs inside divs.

5.2.4 Finding the CSS that matches the element

The main algorithm for this CSS evaluation is the correct matching between the HTML elements, run through one by one, and the CSS rule applied to each one. The problem arises as the <style>and <link >kinds of CSS, unlike the inline CSS, do not appear already with the element in the DOM structure, as the inline CSS does. This means we had to develop an algorithm that would determine which CSS rule set matched the current element. This algorithm is located in a separate file; this is a file where we stored the functions that are needed by all the evaluation techniques which prevented us from having to repeat the same code for each technique.

This algorithm is divided into three different stages:

The first stage is when from all the final CSS structure, explained before, only the css in which the element in evaluation is in the last position of the selector. For example, if the current node to be evaluated was a <p>tag; from a structure with countless CSS rules we would take only those that looked, for example, like these:

body p { } ;
body #maindiv p { } ;
p { } etc.

The second stage involves dealing with rule sets that refer to different CSS. These are identified, separated and then sent into the next stage for classification together with the parents and attributes of the current element, as well as the rule selector in examination.

h2 + p{ text-indent: 0; }
#menu li, p { background-color:yellow; }

are sent as:

h2{ text-indent: 0; }
p { text-indent: 0; }
#menu li{ background-color:yellow; }
#menu p{ background-color:yellow; }

In this next stage a set of attributes is studied and in the end a grade is given to the CSS rule set. First we check to see if there is a class or id match between the different classes or ids in the parents variable and the classes found in the selector in question. The parents variable keeps track of the elements inside which the current element is; and the selector variable keeps the selector of the current css rule set in evaluation. As we can see in the previous examples the cases with an ok result would be classMach true.

This process is repeated for every ruleset selected in the first step of the algorithm and thus why it is so important. For example:

parent variable : body #maindiv .div1 .div2 p a
selector variable 1 : body #maindiv .div2 a — >ok
selector variable 2 : body #maindiv a — >ok
selector variable 3 : body #maindiv .div1.div2 p a — >ok
selector variable 4 : body a — >ok
selector variable 5 : a — >ok
selector variable 6 : body #maindiv .div3 a — >not ok
selector variable 7 : body #footer .div1 .div2 p a — >not ok

Also we test the Type of the element, this is mostly useful for matching when the element is an input. For a type match, if it is defined for the element, the type must be the same as the one in the selector. Otherwise, there is no type match. If the element has no type defined, there is no need to check the type of the element. Next we determine the ratio of the match, for example:

parent variable : body #maindiv .div1 p a
selector variable 1 : body #maindiv .div1 a
This would give a ratio of 4/5.

parent variable : body #maindiv .div1 p a
selector variable 2 : body a
This would give a ratio of 2/5.

This ratio variable allows for a more accentuated distinction between rule sets that are more or less appropriate. In this last example, both selector variable 1 and 2 are correct,

but the first is the best one for that element. The final classification is given according to this:

If element class and current css class match {
    if type match: result is 6 * ratio
    if no type match and no type in element: result is 5 * ratio
    if no type match and type is defined in element but not in current css:
        result is 4 * ratio
    if no type match and type is defined in element and is defined in current css:
        result is 0
}

else if no class match {
    if type match: result is 3 * ratio
    if no type match and no type in element: result is 2 * ratio
    if no type match and type is defined in element but not in current css:
        result is 1 * ratio
    if no type match and type is defined in element and is defined in current css:
        result is 0
}

else: result is 0

This way, in the end, the classification result reflects all the characteristics found during the algorithm execution. A CSS block should have a higher grade if there is a greater number of matchings between it and the current element s characteristics; such as element name, element class or type and elements' parents.

5.2.5 Technique Error Reporting

Each time a technique is executed a specific set of information used for reporting if an error was or not detected in the evaluation process is produced. This specific set of attributes are described in the following table:

Elements	Contenct	Description	Grup
aux["url"]	url	URL of the Webpage to be evaluated	Source
aux["file"]	file	File name where the error occurs (0 if main HTML file / URL if any external CSS file)	Source
aux["src"]	originalCss	CSS code line where the error occurs	Source

aux[”linenumber”]	lineN	Line number of the file where the error occurs	Source
aux[”technique”]	”c21”	Technique that detected the error	Result
aux[”result”]	out	Result of the evaluation (FAIL, WARNING or PASSED)	Result
aux[”resultType”]	outType	Type of the Error, detailed ahead	Result
aux[”repair”]	originalCss	Field to store the repair realized over the src;	Result
aux[”position”]	pos	Element (Tag and Attributes) where the evaluation was triggered	Internal
aux[”elem”]	elem	Element where the error occurred	Internal
aux[”index”]	arrayIndex	Internal identifier to be used in the repair process;	Internal

Table 5.3: Description of the Error Reporting Structure

In table 5.3 we can also see that the attributes were organized into 3 different groups. The first are the attributes used for locating the error on the files; the second group are attributes used for identifying what kind of error was found; and the last group consists of attributes used for internal use in the repair and evaluation tool.

The definition of these attributes is crucial to the repair stage specifically: the technique that generated the result; the outcome of the evaluation; the outcome type, since there are several different types of fails; the repair attribute, that stores the css used and that will be repaired; and finally the array index that allows for an easier access to the CSS structure. Both the technique and the outcome type together are the ones used for the matching of the error with the solution. For example, an error identified in an inline CSS, cannot be repaired the same way as as error identified in a linked CSS. This is detailed ahead in the repair section of this document.

The aux[”resultType”] attribute is highly related to our interpretation of the techniques. In general its values will vary in the following way:

- 0 and 20 for passes. 0 for linked and style CSS and 20 for inline CSS. Whenever this value is returned the technique was verified and does not need a repair.
- 1 is returned if a CSS rule was not found for the current element.
- 2 variations are used to specify situations in CSS found inline; for example if font-size property is found inline and has a wrong unit a 23 is returned. If a line-hight property has value too low a 211 is return 212 if to high and 23 if wrong unit.
- 3 and all the following numbers are used to detail other types of errors in linked and style CSS.

5.3 CSS and HTML evaluation coming together

QualWeb, was planned to run two separate evaluations, the HTML and the CSS, in order to save execution time, whenever the developer did not want to run the CSS evaluation. So as we finished the CSS evaluation process, CSS evaluation and HTML evaluation were two different things that run separately. Also HTML technique run differently than CSS techniques. While CSS techniques were planned from the beginning in an optimized way, the index file which run all the DOM elements and triggered the CSS .js files, HTML techniques, individually run all the DOM.

This way, a common evaluation process would run all the HTML techniques and afterwards the complete CSS evaluation. So when we finished the CSS evaluation one objective was to merge HTML and CSS evaluations so that HTML techniques could avoid code repetition.

For this, the index file created for the CSS evaluation, where all the iteration of the DOM structure is done and from where techniques, are triggered, was extended to include HTML techniques. The purpose of this part of the work was not only to optimize the code reducing execution time but also to make sure HTML techniques had the same return format as the CSS techniques. We wanted to use our experience, obtained during the development of the repair module, to leave HTML techniques already prepared for future repair development.

5.3.1 Retrieving element's information

For this optimization to be implemented in the HTML evaluation, all implemented HTML techniques needed to be changed. So we started with every HTML technique implemented and reformulated it. The main challenge was that some of these techniques require parents or sibling elements which were easy to verify running the whole DOM structure but impossible if the technique only received the element. This was solved by using the same strategy used for the CSS techniques, described in subsection 5.2.3.

So, first, we used the index's file algorithm and added the HTML technique triggering. Next in each individual technique we removed the DOM iteration and added the iteration through siblings and parents. Many HTML techniques required the evaluation to test if direct parent or sibling had a specific attribute. For example technique H44 has the following test procedure:

"For all input elements of type text, file or password, for all textareas and for all select elements in the Web page:

1. Check that there is a label element that identifies the purpose of the control before the input, textarea, or select element

2. Check that the for attribute of the label element matches the id of the input, textarea, or select element

3. Check that the label element is visible. ”

This technique will be triggered by the input element, although it also requires that we get its parents. This is mandatory in order to obtain the label which has: `label[attributes][for] == input[attributes][id]` (or class).

Although we needed to add and remove somethings, we wanted to alter the minimum possible. In the end, no HTML technique needed to be completely redone. The objective was to keep their algorithms as close to what they were as possible, since they had already been tested.

5.3.2 Altering the HTML techniques

After this, techniques shown in the following table 5.4 were updated. From all the techniques implemented, only two were not altered until the submittal of this report, these are H65 and H74 referring to

- h65 - Using the title attribute to identify form controls when the label element cannot be used
- h74 - Ensuring that opening and closing tags are used according to specification

This happened because these two techniques were much more complex than the others and thus required more time to be implemented. Overall changes applied, were in error report structure and adding parents and brothers search. All DOM processing was removed from these files into the index .js file from where techniques are triggered.

Changes in the error report structure, included adding some new attributes, in order to make HTML techniques return the same elements as CSS techniques. Before this work, HTML techniques returned only the attributes:

- Technique - The technique that evaluated this occurrence.
- Result - With values passed, failed or warning
- Source - With the source for the problem
- URL - The page url
- Type - Value to indicate the type of the error. Although techniques were still not refined not separate different types of results.

- Elem - The element where the error occured

Previously, since each technique had access to the whole DOM document, obtaining sibling elements and parents was no big problem. But with these changes, and since form each element it is impossible to access its parents, a new approach had to be developed. At this stage it was not a problem, since we used the same process as we did for the CSS, which is described in subsection 5.2.3.

The following table introduces some information on the implemented HTML techniques that were updated:

Techniques	**Description**
h2	Combining adjacent image and text links for the same resource
h24	Providing text alternatives for the area elements of image maps
h25	Providing a title using the title element
h27	Providing text and non-text alternatives for object
h30	Providing link text that describes the purpose of a link for anchor elements
h32	Providing submit buttons
h33	Supplementing link text with the title attribute
h35	Providing text alternatives on applet elements
h36	Using alt attributes on images used as submit buttons
h37	Using alt attributes on img elements
h39	Using caption elements to associate data table captions with data tables
h44	Using label elements to associate text labels with form controls
h45	Using longdesc
h46	Using noembed with embed
h50	Using map to group links
h53	Using the body of the object element
h57	Using language attributes on the html element
h59	Using the link element and navigation tools
h64	Using the title attribute of the frame and iframe elements
h67	Using null alt text and no title attribute on img elements for images that AT should ignore
h71	Providing a description for groups of form controls using fieldset and legend elements
h76	Using meta refresh to create an instant client-side redirect
h89	Using the title attribute to provide context-sensitive help
h91	Using HTML form controls and links
h93	Ensuring that id attributes are unique on a Web page

Table 5.4: Description of the altered HTML Techniques

In general these techniques involve searching for specific attributes inside the tag, or finding a specific father or even a father with a specific attribute. For example technique c24 needs the image to have an alt text attribute.

5.3.3 Return Results in a different perspective

Previously to this work QualWeb kept the techniques' results in a structure organized by technique:

{ [H15: { Object; Object; Object}]
[H44: { Object}]
[C121314 : { Object; Object; Object}]
[C17 : { Object; Object; Object}]
[C21 : { Object}] }

This worked just fine, as results were afterwards presented by technique. But as this work came to an end, interesting thoughts for the future of QualWeb started to appear and criteria based evaluation is now being considerate.

We took the opportunity during the development of the integration between CSS and HTML evaluation, to make the necessary preparations for this change. Since the DOM structure was now being run through element by element for both the HTML and the CSS, we just needed to create a new structure that would be filed with HTML structure in mind the following way:

{ Object; Object; Object; Object; Object; Object; Object; Object; Object; Object;}

Since information about the techniques is already inside each element, thanks to the return format of each technique, no information is lost in this process.

For now, these two structures were both left in, so they can be used depending on the team's needs.

5.4 Testing the CSS implementation

During and After the process of implementing the CSS techniques we developed a set of tests to verify these techniques. In the beginning we developed small simple pages, one for each technique that contained several possible types errors that should be identified by the techniques. They were based on the examples given by the WCAG 2.0 CSS techniques,

but sometimes they were extended to include several other situations.

As soon as the implementation was stable, we understood that these small and simple tests were indeed too simple and decided to create a single file. We wanted to have a web page of considerable size, to simulate the confusion of a real Web page. This indeed proved to be the right solution since we used it to locate some situations, where the algorithms needed to be adjusted. Specifically the algorithm that connected each element with their corresponding CSS.

Again when this reached a stable point in the development we started to evaluate a list of real Web pages. This was a major reality check, since although our Web page was large and more complex than the simple individual tests, it did not even compare to the complexity and confusion a real Web page is. From this we were able to refine our algorithms and also to make some code optimizations.

In the end we had algorithms that went through several stages of optimizations.

Chapter 6

Repairing the Web

In this chapter we describe the development of the repair tool. Here are the details of how we interpreted all the different techniques, from the repair point of view, as well as how we made de connection between the evaluation and the repair tool.

During this process we faced some setbacks, making us change our objective. We changed from complete automated repairs and return of altered files, to a repair suggestion approach.

In the end of this chapter, we proceed to repair actual Web pages, in order to make a self assessment of our repair suggestions.

6.1 Analysis

Before proceeding with the repairs, we started by grouping techniques in groups according to difficulty of repairs. This way we could use easier and more direct techniques, to do our first experiments with the tool and build it from there.

This way we divided techniques into the following groups according to characteristics:

- Techniques C7, C12, C13, C14, C17 and C21, were classified as techniques of direct repair. They were given higher priority, since we could use them to structure the repair algorithm. Also these techniques are the ones that could be 100% automated and do not need human verification.
- Techniques C15, C23 and C25 were also classified as techniques of direct repair. However all require color suggestions which we cannot choose automatically, since the developer may want to choose particular colors as alternatives. This way we can only make suggestions.

- Techniques C8, C9 and C19 were the only ones that required altering the DOM by altering specific tags or its content. When we abandoned the fully automated approach, they were inserted in the previous group.
- Techniques C20, C24, C26 and C28 - All regard widths of containers and so could be considered together for repairing. These were not to be considered in the first group since widths cannot simply be converted from absolute to relative units.
- Finally, techniques C6, C18, C22, C27, C29 and C30 were not implemented in the evaluation and will not be considered in this chapter.

6.2 Repair Process

The evaluation process returned a structure filled with every return of each technique. To begin the repair procedure, the repair tool must receive this structure.

From here, for every one of these results, if the outcome of the evaluation is a ”failed” or ”warning” then the repair process is activated.

6.2.1 Error Types

Repairs are applied according to the return type, returned by the evaluation in each technique. Table 6.1 has a conversion between types of error and the acronyms used in table 6.2.

Return Type	Description	Acronym
0	Signals a Pass. No error was found in the CSS for this element	P
1	Signals a Fail or a Pass, caused by a lack of CSS	NCSS
20	Signals a Pass. CSS inline has property correctly applied and makes the technique pass	IP
21 22 23 211 212	Signal different types of errors in the inline CSS	IE
3 4 5	Signal different types of errors in the CSS found	E

Table 6.1: Outcome Type Description

6.2.2 NCSS type of Errors

TYPE NCSS errors or passes, are errors or passes derived from the fact that there is no CSS rule for that element.

Lack of CSS can cause a pass or a fail, depending of the technique, for example: In techniques C12 C13 and C14, whenever there is no CSS found, text in paragraphs is resizable by definition. This results in a pass In techniques C2 however, whenever there is no CSS found, text in paragraphs has line-heigh with value 1.0 em. Meaning that by definition, paragraphs line-height is lower than 1.5, the minimum possible value. This results in a fail.

Repairs consist of the suggestion of the insertion of a new CSS rule.

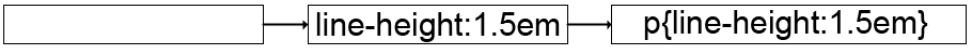

Figure 6.1: An example of a Type NCSS Repair

6.2.3 IE and E type of Errors

- An IE problem is caused by an error inline. Repairs for these types of errors, generally involve correcting the style attribute for the element. This would be a type of repair that in a fully automated tool, would require a change in the DOM.
- An E problem likewise is caused by an error but in one of the remaining types of CSS. Repairs for these types of errors, generally only require CSS to be changed.

Figure 6.2: An example of a Type IE Repair

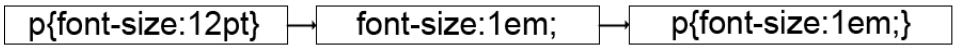

Figure 6.3: An example of a Type E Repair

6.2.4 Errors Identified by Technique

Tech	Type	Description of the Error
6	–	——
c7	P	Target Element has span child, a description and a CSS for hidding the description
	NCSS	Target Element has span child but no CSS for span element some description

	IP	Target Element has span child and an incomplete CSS for span element, with all the properties inline some description
	IE	Target Element has span child and a incomplete CSS for span element, with properties inline some description
	E	Target Element has span child and a incomplete CSS for span element some description
	E	No child span found following the Target Element some description
	E	Target Element has span child, but no description and no CSS were found
c8	P	Found no bad usage of space characters
	NCSS	Found bad usage of space characters in text and no CSS rule
	IE	Found bad usage of space characters in text and CSS inline
	E	Found bad usage of space characters in text and CSS
c9	P	Image found is not a decorative image
	NCSS	Image found is a decorative image
c121314	P	Correct font-size unit: "em","% "
	NCSS	Element without CSS defined - font-size resizable by definition
	IE	Absolute font-size in inline style attribute
	E	Absolute font-size in CSS property
	P	Element has CSS rule but no font-size property
c15	P	Anchor or input with hover or focus property and color or background color
	NCSS	Anchor or input without CSS
	IE	Anchor or input with either hover or focus property inline
	E	Anchor or input with either hover or focus property
c17	P	Label with correct font-size: "em","% "
	NCSS	Label without CSS defined - font-size resizable by definition
	IE	Absolute font-size in inline style attribute
	E	Absolute font-size in CSS property
c18	–	——
c19	P	CSS defined and no Error in text
	NCSS	No CSS defined and incorrect usage of space characters in text

	P	No CSS defined and no Error in text
	IE	Found inline CSS and incorrect usage of space characters in text
	IE	Text either justified or aligned to the center by style attribute inline
	E	Text either justified or aligned to the center
	E	Found CSS rule and incorrect usage of space characters in text
c20242628	P	Target Element has width property and relative mesure
	NCSS	No CSS rule was found div with width unspecified
	IE	Absolute unit found for Target Element in inline style attribute
	E	Absolute unit found for Target Element
	P	Found CSS but no width property
	E	Element has ”auto” width
c21	P	Element has line height property with ”%” or ”em”unit and correct values
	NCSS	Element has no CSS rule
	IE	CSS in style attribute has line height with value lower than 1.5`em` or 150 `%`;
	IE	CSS in style attribute has line height with value greater than 2.0`em`. or 200`%`
	IE	CSS in style attribute has line height with absolute unit
	E	Element has line height with value lower than 1.5`em` or 150`%`
	E	Element has line height with value greater than 2.0`em` or 200`%`
	E	Element has line height with absolute unit
	E	Element has no line-height property”
c22	–	——
c23	P	Element has no color defined
	NCSS	Element has no CSS rule
	IE	Element has Background Color in inline CSS attribute
	IE	Element has Text Color in inline CSS attribute
	IE	Element has both Background and Text Colors in inline CSS attribute
	E	Element has Background Color
	E	Element has Text Color
	E	Element has Background Color and Text Color
c25	P	Element has no color defined

	NCSS	Element has no CSS rule
	IE	Element has Background Color in inline CSS attribute
	IE	Element has Text Color in inline CSS attribute
	IE	Element has both Background and Text Colors in inline CSS attribute
	E	Element has Background Color
	E	Element has Text Color
	E	Element has Background Color and Text Color
c27	–	——
c29	–	——
c30	–	——

Table 6.2: Outcome Type Description

6.3 Repairing CSS Return Types

Currently QualWeb's repair module, given the 6.1 table approach to return types from the evaluation, instead of repairing the files themselves, stores a string with the recommended steps the developer should follow, in order to make his/hers Web page compliant with each of the WCAG 2.0 technique.

This helps liberalize the process. Now instead of forcing our repairs on the developer, we point out what is wrong, we explain how and why it is wrong and we inform how our approach is correct as well as inform other ways that also could be correct by saying something equivalent to: "NOTE: You can change this values for font-size, just make sure you use relative font-sizes".

We hope that with these sort of descriptions we can put ourselves closer to the developer by saying: "just think about it".

Following this line of though here we present a table with the different repairs and notes applied to each error type:

Tech	**Type**	**Description of the Error**
c6	–	———
c7	P	———

	NCSS	1 - This target element already has a span child but if this span is being used for another reason, make sure you add a new span tag inside the target element with a description to be hidden. For example: REPLACE this text with your own 2 - Hide this new span tag by adding the following css: a .hiddingLinkDesc { height: 1px; width: 1px; position: absolute; overflow: hidden; top: -10px } NOTE: You can change the class value just make sure, both have the same class or id [1]
	IP	———
	IE	1 - This target element already has a span child but if this span is being used for another reason, make sure you add a new span tag inside the target element with a description to be hidden. For example: REPLACE this text with your own 2 - Hide this new span tag by adding the following css: a .hiddingLinkDesc { height: 1px; width: 1px; position: absolute; overflow: hidden; top: -10px } NOTE: You can change the class value just make sure, both have the same class or id
	E	1 - This target element already has a span child but if this span is being used for another reason, make sure you add a new span tag inside the target element with a description to be hidden. For example: REPLACE this text with your own 2 - Hide this new span tag by adding the following css: a .hiddingLinkDesc { height: 1px; width: 1px; position: absolute; overflow: hidden; top: -10px } NOTE: You can change the class value just make sure, both have the same class or id

[1]The suggested repairs are very similar for different types of error, since we have no way of knowing if the existing span child is the one we were looking for, or just a non related span child. Also for this reason c7 is presented in the interface as a warning and not as a fail.

	E	1 - Add a span tag inside the target element with a description to be hidden. For example: REPLACE this text with your own 2 - Hide this new span tag by adding the following css: a .hiddingLinkDesc { height: 1px; width: 1px; position: absolute; overflow: hidden; top: -10px } NOTE: You can change the class value just make sure, both have the same class or id
	E	1- No direct text description was found for this element. Make sure it does not need one. 2 - Add a span tag inside the target element with a description to be hidden. For example: REPLACE this text with your own 2 - Hide this new span tag by adding the following css: a .hiddingLinkDesc { height: 1px; width: 1px; position: absolute; overflow: hidden; top: -10px } NOTE: You can change the class value just make sure, both have the same class or id
c8	P	———
	NCSS	1 - Replace only the text that currently contains the character(s) with this: " text " 2 - If the text has any words that were spaced using the character, surround them with a new 3 - Add the following css rule for this element: .spacedPar { letter-spacing="Xem" } NOTE: You can change this class value or choose another way to present the text, just make sure you remove the nbsp characters [2]
	IE	Replace only the text that currently contains the character(s) with this: " text " 2 - If the text has any words that were spaced using the character, surround them with a new 3 - Add the following css rule for this element: .spacedPar { letter-spacing="Xem" } NOTE: You can change this class value or choose another way to present the text, just make sure you remove the nbsp characters

[2] The suggested repairs are also similar since we have to add a new element, the new span tag, and style it with the appropriate letter-spacing.

	E	Replace only the text that currently contains the character(s) with this: ” text ” 2 - If the text has any words that were spaced using the character, surround them with a new 3 - Add the following css rule for this element: .spacedPar { letter-spacing=”Xem” } NOTE: You can change this class value or choose another way to present the text, just make sure you remove the nbsp characters
c9	P	———
	NCSS	1 - Make sure this is a real decorative image and not just an image with an empty alt attribute 2 - If this is not a decorative image, add alt value and finish the repair; If this is a decorative image, replace the decorative img tag with 3 - Add the following CSS rule: #imgscrname { background:url(imgscr); no-repeat display: inline-block; height:20px; width:20px;} NOTE: Adjust the placement and the size of this image, just make sure you add the image through the CSS.
c121314	P	———
	NCSS	———
	IE	1 - Replace the current inline CSS with: font-size: Xem NOTE: This is just a conversion from the absolute unit, you can adjust this value just keep using relative units
	E	1 - Replace the current CSS property with: font-size: Xem NOTE: This is just a conversion from the absolute unit, you can adjust this value just keep using relative units
	P	———
c15	P	——— 1 - Add the following CSS property(ies): elem:hover{background-color: #DCE7F6;color:#000000} elem:focus{ background-color: #DCE7F6; color:#000000}
	NCSS	NOTE: You can change the colors to the ones you want, just make sure they follow the contrast ratio also required by WAG 2.0

	IE	1 - Add the following CSS property: elem:hover{background-color: #DCE7F6;color:#000000} elem:focus{ background-color: #DCE7F6; color:#000000} NOTE: You can change the colors to the ones you want, just make sure they follow the contrast ratio also required by WAG 2.0 [3]
	E	1 - Add the following CSS property: background-color: #DCE7F6; color:#000000 elem:hover{background-color: #DCE7F6;color:#000000} elem:focus{ background-color: #DCE7F6; color:#000000} NOTE: You can change the colors to the ones you want, just make sure they follow the contrast ratio also required by WAG 2.0 [4]
c17	P	———
	NCSS	———
	IE	1 - Replace the current inline CSS with: font-size: Xem NOTE: This is just a converted value you can change it but keep in mind to always use relative font-sizes.
	E	1 - Replace the current CSS property with: font-size: Xem NOTE: This is just a converted value you can change it but keep in mind to always use relative font-sizes.
c18	–	———
c19	P	———
	NCSS	1 - Replace only the text that currently contains the character(s) with this: ”text” 2 - If the text is not empty, surround it with a new 3 - Add the following css rule for this element: .alignedText { text-align: right } NOTE: You can change this class value or choose another way to present the text, just make sure you remove the nbsp characters and use the align property correctly
	P	———

[3] In this case only one is shown, depending on which one is missing for the current element

[4] Only two of these three rules are shown in the interface, one for hover and one for focus, depending on which one is missing for the current element

	IE	1 - Replace only the text that currently contains the character(s) with this: ”text” 2 - If the text is not empty, surround it with a new 3 - Add the following css rule for this element: .alignedText { text-align: right } NOTE: You can change this class value or choose another way to present the text, just make sure you remove the nbsp characters and use the align property correctly
	IE	1 - Remove the ”text-align:” property
	E	1 - Remove the ”text-align” property
	E	1 - Replace only the text that currently contains the character(s) with this: ”text” 2 - If the text is not empty, surround it with a new 3 - Add the following css rule for this element: .alignedText { text-align: right } NOTE: You can change this class value or choose another way to present the text, just make sure you remove the nbsp characters and use the align property correctly
c20242628	P	——
	NCSS	——
	IE	1 - Replace the old inline CSS for this Target Element with: width: Xem NOTE: This is just a conversion, you can change this value just remember to use relative font sizes for width containers NOTE2: No unit was defined so px is used by the browser by omission [5]
	E	1 - Replace the old CSS for this element with : width: Xem NOTE: This is just a conversion, you can change this value just remember to use relative font sizes for containers’ width NOTE2: No unit was defined so px is used by the browser by omission
	P	——
	E	NOTE : This is marked as a warning since ẅidth: autod̈oes not fit in any of the mentioned techniques and since it is not a absolute unit either.
c21	P	——
	NCSS	——
	IE	1 - Replace the old inline CSS for this element with: ”line-height: 150%”

[5]NOTE2 will only be used whenever a unit for width property is not defined.

	IE	1 - Replace the old inline CSS for this element with : ”line-height: 200%
	IE	1 - Replace the old inline CSS for this element with: ”line-height: Xem” NOTE: You can change this line-height value, just remember to use relative units and mantaining heigth between 150% and 200% or 1.5em and 2.0em
	E	1 - Replace the old CSS for this element with: ”line-height: 150%”
	E	1 - Replace the old CSS for this element with: ”line-height: 200%”
	E	1 - Replace the old CSS for this element with: ”line-height: Xem” NOTE: You can change this line-height value, just remember to use relative units and mantaining heigth between 150% and 200% or 1.5em and 2.0em
	E	1 - By omission line-heigh is not compliant with W3C values, so you need to add to the CSS of the target Element the property: line-height:1.7em
c22	–	——
c23	P	——
	NCSS	——
	IE	Replace the Original inline CSS with this one: ”cssRuleToInsert” [6]
	IE	Replace the Original inline CSS with this one: ”cssRuleToInsert”
	IE	Replace the Original inline CSS with this one: ”cssRuleToInsert”
	E	Replace the Original CSS with this one: ”cssRuleToInsert”
	E	Replace the Original CSS with this one: ”cssRuleToInsert”
	E	Replace the Original CSS with this one: ”cssRuleToInsert”
	E	Replace the Original CSS with this one: ”cssRuleToInsert”
c25	P	——
	NCSS	——

[6]The same CSS ,without the background and color properties

	IE	Replace the Original inline CSS with this one: ”cssRuleToInsert” [7]
	IE	Replace the Original inline CSS with this one: ”cssRuleToInsert”
	IE	Replace the Original inline CSS with this one: ”cssRuleToInsert”
	E	Replace the Original CSS with this one: ”cssRuleToInsert”
	E	Replace the Original CSS with this one: ”cssRuleToInsert”
	E	Replace the Original CSS with this one: ”cssRuleToInsert”
	E	Replace the Original CSS with this one: ”cssRuleToInsert”
c27	–	——
c29	–	——
c30	–	——

Table 6.3: Description of the Repairs applied

6.4 Setbacks

In the beginning of the implementation of the repair module we were aiming at a different approach. We wanted to fully automatically repair Web pages, including the HTML and CSS files. But in the end this proved an ineffective way of approaching the problem of accessibility repair. This chapter describes the process followed, what was done and why this line of work was dropped.

6.4.1 Types and Repairs

Previously repairs were applied according to what is specified in table 6.4.

Tech	Type	Description
c6	–	——
c7	NCSS	A standard description is added to the DOM and new CSS rule, to hide the textual description, is created
	IE	The new CSS rule for ”span” is added inline
	E	Since the span CSS is not complete, the remaining properties are added.

[7] The same CSS ,without the background and color properties

	E	The span tag and the standard description are added to the DOM, and a new rule for span are created
	E	The standard description is added
c8	NCSS	A new rule for the element is created with the letter-spacing property
	IE	The letter-spacing property is added to the style attribute
	E	The property letter-spacing is added to the CSS found
c9	NCSS	The img tag is removed from the DOM, the image is put in a span tag and a CSS rule is added for that specific img
c121314	IE	The value of the current font-size is obtained and converted into `em` units and replaced inline
	E	The value of the current font-size is obtained and converted into `em` units and replaced
c15	NCSS	Insert a color or background color property into the a newly created CSS rule
	IE	Insert a color or background color property into the inline CSS
	E	Insert a color or background color property into the existing CSS
c17	IE	A unit conversion into ”em” unit is done and replaced over the old CSS inline
	E	A unit conversion into ”em” unit is done and replaced over the old CSS
c18	–	——
c19	NCSS	An align-center property is added to a new CSS rule
	IE	An align-right property is added to a new CSS rule
	IE	An align: center property is added to the inline CSS rule
	IE	An align-right property is added to the inline CSS rule
	E	An align: center property is added to the existing CSS rule
	E	An align-right property is added to the existing CSS rule
c20242628	IE	A unit conversion into ”em” unit is done and replaced over the old one inline
	E	A unit conversion into ”em” unit is done and replaced over the old one in the existing CSS
c21	IE	The old value is replaced inline with 150% or 1.5em depending on what the original unit was
	IE	The old value is replaced inline with 200% or 2em depending on what the original unit was
	IE	The old inline value is converted to em inline and adjusted depending on the value

	E	The old value is replaced with 150% or 1.5em depending on what the original unit was
	E	The old value is replaced with 200% or 2.0em depending on what the original unit was
	E	The old value is converted to em inline and adjusted depending on the value
c22	–	——
c23	IE	Remove inline background color
	IE	Remove inline text color
	IE	Remove inline both background color and text color
	E	Remove background color
	E	Remove text color
	E	Remove both background color and text color
c25	IE	Remove inline background color
	IE	Remove inline text color
	IE	Remove inline both background color and text color
	E	Remove background color
	E	Remove text color
	E	Remove both background color and text color
c27	–	——
c29	–	——
c30	–	——

Table 6.4: Description of the Repairs initially planned

6.4.2 Returning Files

In this initial line of thought the objective was to return, as a result, all the repaired files connected to the Web page. Repairs were applied not over the HTML itself, but over the DOM, since it has an easier structure to manipulate. This altered DOM is in the end converted into a string and then written into a .html file.

For the external CSS, a new file with the same name as the original, was created for each external .css retrieved, creating n files to return. This covers repairs of type E applied to external files. Following this, a completely new external file needed to be created to keep CSS rules created anew during the repair process (NCSS repairs). Since they are created instead of altered they are not directly replaceable into existing structures.

The last step is to build the .html file. For this, all nodes of the DOM structure are read and translated into a HTML notation, except:

- After the last <linked >tag a new one is added with the link for the newly crated .css file with the NCSS repairs.

- At the <style >tag, which content is fully replaced by the CSS structure build in the pre-processing. The structure, which maintains all the comment blocks from the original content and already holds the E repairs.

Since IE repairs imply changes that need to be applied directly on the DOM structure, these were translated into HTML notation rather easily, along with the remaining tags.

6.4.3 Why this was left behind

After all these repairs being implemented, we started to view them from a different perspective, what does this mean? How can we blindly say to the developer: ”this is how you have to develop your Webpage”, without having a global picture of the page and its semantics? Returning a HTML file that would most likely be totally different from what the developer intended in the first place, would be the best way for our tool to loose credibility. This was the reason we left the 100% automatic repairs behind and started to focus on the repairs suggestion approach.

6.5 Evaluating the Repair Process

In order to validate our repair suggestions we proceeded to a manual repair. For this, and since this requires some time and is a somewhat cumbersome task, we selected a couple of the most visited webpages, identified by alexa.com.

6.5.1 Test pages

From the opened Webpage we clicked the ”save as” option and the necessary files would be downloaded. These were placed online on our server, in order to be evaluated by QualWeb directly from the website.

Youtube

Starting with the Youtube page, QualWeb evaluator identified 179 fails, divided through techniques c9, c15, c19 and c21. These techniques concern decorative images, images with null attribute, missing background decorations of input and anchors, use of nbsp characters and the line-height properties. Also 118 warnings were found in technique c7, concerning anchors without a description.

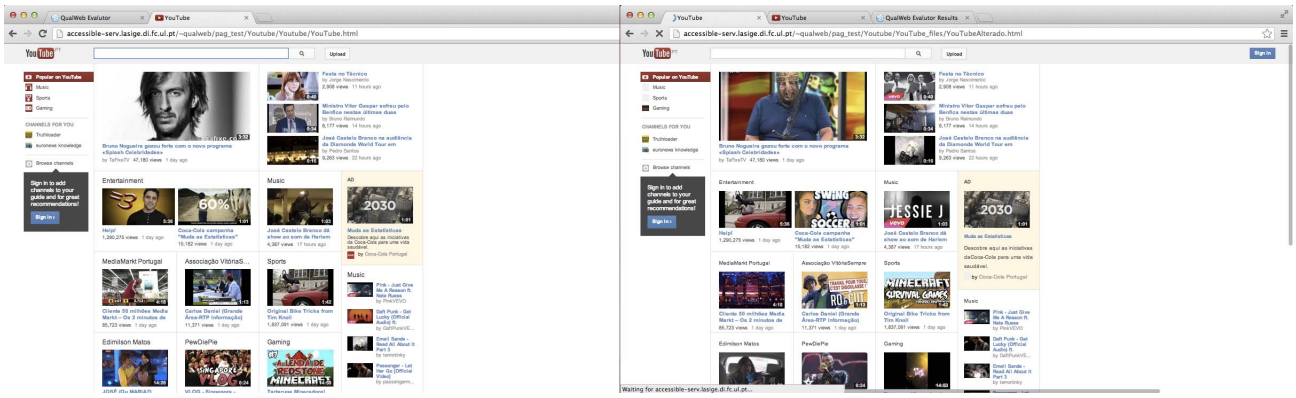

Figure 6.4: Youtube Webpage before (left) and after (right) repairs

Repairs applied to Youtube page did not change anything.

Wikipedia

Regarding the Wikipedia Web page, QualWeb identified 371 fails, divided by techniques c9, c15, c20242628 and 329 warnings in technique c7 and c20242628. Fails were found in decorative images, missing background decorations of input and anchors, use of nbsp characters and line-height proper tie with wrong values. Warnings were found in anchors, missing descriptions, and widths using absolute units.

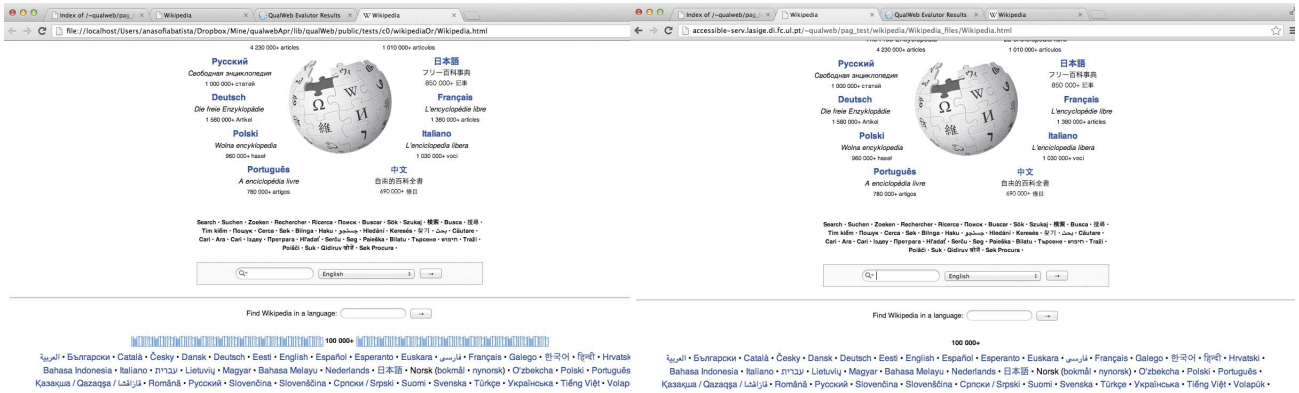

Figure 6.5: Wikipedia Webpage before (left) and after (right).

In the repair version, we can see that images of books on the bottom of the screenshot have disappeared. Probably the page would need extra work to make sure such images are maintained. Everything else is unchanged.

Amazon

The following page that we evaluated and repaired was the Amazon Webpage. Our tool identified 48 fails and 317 warnings in techniques c9, c20242628, c21, c23, c25 and c7 c20242628, c23 and c25 respectively.

The biggest problem with this Webpage was that all images in content were marked as decorative, since they had null alt value, providing false positives. Other fails consisted of widths with wrong units, line-heigh with wrong values and use of colors in main content. Warnings were anchors without descriptions, width with absolute units and possible containers in main content with background or text colors.

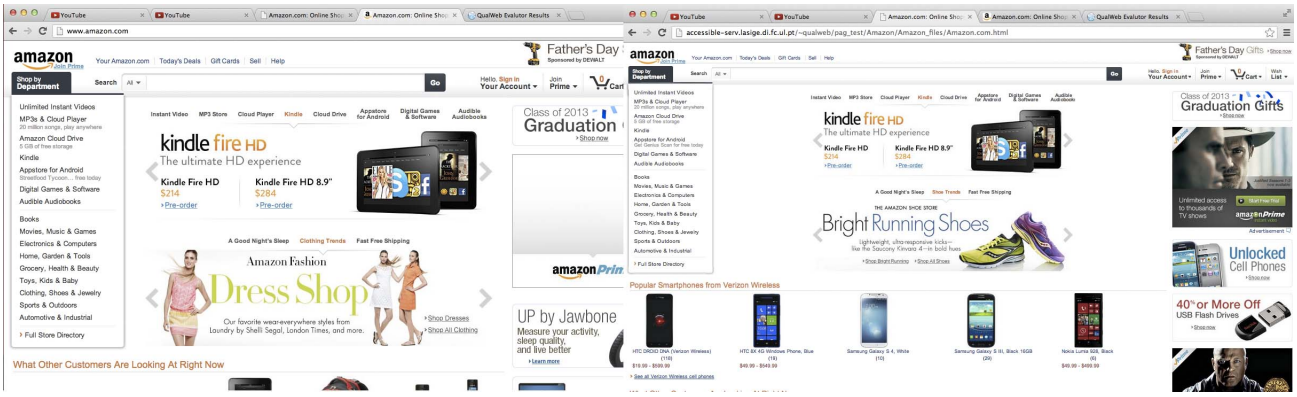

Figure 6.6: Amazon Webpage before (left) and after (right) repairs.

The repaired page, seems to have reduced the zoom of the page, although none of the changes made were directly related to that property. This can mean that some changes can influence other non related aspects of the page.

Yahoo

The last evaluated page was the Yahoo Webpage. Results were, 377 fails in techniques c8, c9, c15, c19 and c21, and 296 warnings in techniques c7, c23 and c25. Fails concerned, texttt& nbspcharacters, decorative images in the HTML and unspecified line-height. Warnings concern anchors without descriptions and div background and text colors.

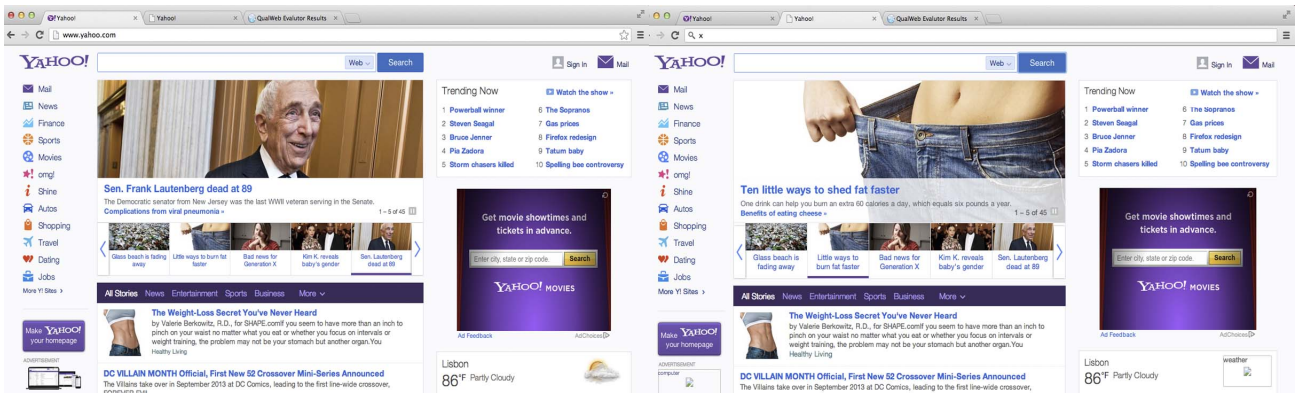

Figure 6.7: Yahoo! Webpage

With this page we had some problems linking images and css files. Because of this, in the right image on figure 6.7 we can see, on the bottom left corner, an image missing. Anyway, this does not have anything to do with the applicability of repairs.

6.6 Critique

For this section we acted as the developer of the Webpage by applying each repair solution one by one by hand. The objective was to understand how applicable these suggested repairs were. The first thing that came into view, was that these Webpages were of considerable size and thus, the number of elements was significant.

Apart from all this, we realized that after applying all these changes, the developer would need to make some new adjustments, as some items were misplaced or disappeared. Anyway, an expert user, or the actual developer, or developer team, would have more experience in order to repair these pages accordingly. One thing that we noted, is that repairing these webpages was a cumbersome task due to their size and the amount of code that had to be verified by hand.

Technique c7

Is one of the most triggered techniques. Its repair consists of adding a span to the anchor element, inside this span adding a description for the link and in the end styling this description to not be presented.

This is not a straightforward approach since in reality some anchors already have span tags inside for other reasons. We can say that our technique makes all these recommendations but we still feel these are not very forthright.

Technique c8

Recommends removing nbsp characters and surround the spaced words with a span tag whose style includes a letter-spacing property. Unlike the previous technique this one was pretty simple to repair.

Technique c9

Asks the developer to add decorative images through CSS, instead of through HTML. Our repair steps stated that in order to solve this, image tags should be replaced by span tags, which would then be styled so that the image appears inside as a background.

To apply this technique, we faced some challenges with linking images, although this was not related directly to the repairs. Overall images were correctly replaced by CSS but further testing would be advisable to ensure this.

Techniques c12 c13 c14 and c17

These techniques are very straightforward to repair since they only require font-sizes to be in relative units. In our suggestions, we also present, the developer, with a direct conversion that he may or may not use, depending if it is adequate. Sometimes repair suggestions have a `nan` value for font-sizes, and this means an error in the conversion process. Further refining would be advisable.

Technique c15

Another of the most triggered techniques and also the least elementary to repair. Every anchor or input element must, whenever the element is interacted with, change presentation. Repair steps range from adding a new CSS, to adding just a color change.

Repairing this, required locating the elements, one by one, and finding the appropriate color for background and text. During this process we identified that the implementation of this technique, would benefit, from a more detailed study of the ”element:hover” or ”element:focus” CSS properties.

For example, an element which increases or reduces its width whenever interacted with, would pass since it verifies the techniques:

"... demonstrate how visual appearance may be enhanced via style sheets to provide visual feedback when an interactive element has focus or when a user hovers over it using a pointing device ... "

Technique c19

Much like technique c8, this one detects sets of `& nbsp` characters used to simulate the alignment of text to either right or left. It also verified if the text is incorrectly justified or aligned to the center.

The repair procedure is also quite simple, the developer only has to remove these characters and surround the whole sentence with a span element whose style would accurately align it.

Techniques c20 c24 c26 c28

Turned into a single technique, are supposed to verify widths of containers. This has the trickiest repair suggestions; the user cannot simply be presented with a conversion since 100% of the time these conversions are wrong.

The best thing here, would be to change the repair suggestion so that it would simply advise the developer to use relative units for containers, instead of presenting a suggested value that is most of the times inappropriate.

Technique c21

Refers to the line-height property and its repair is also really easy, in the simplest of cases the developer needs only to add the following CSS rule: `bodyline-heigh:1.7em` and repair other line-heigh specifications that override this one.

Techniques c23 and c25

Also have straightforward repairs, the user only needs to remove all color styling in containers in the main content ares. However, for technique c23, the user still has to be advised to specify borders in layout as the technique description states.

Chapter 7

QualWeb Interface

This chapter explains the process of enhancing QualWeb's interface. We compare before and after versions of the Web page and explain why we applied these changes. After all the development is finished, we proceed with user testing of this interface. In this chapter we detail this experience as well as the results we obtained.

7.1 Previous User Interface

As we have mentioned before, parallel to this work a user interface for QualWeb was developed. After a developer has clicked the evaluation button, the page for presenting the results is shown. This page follows a structure depicted in figure 7.1.

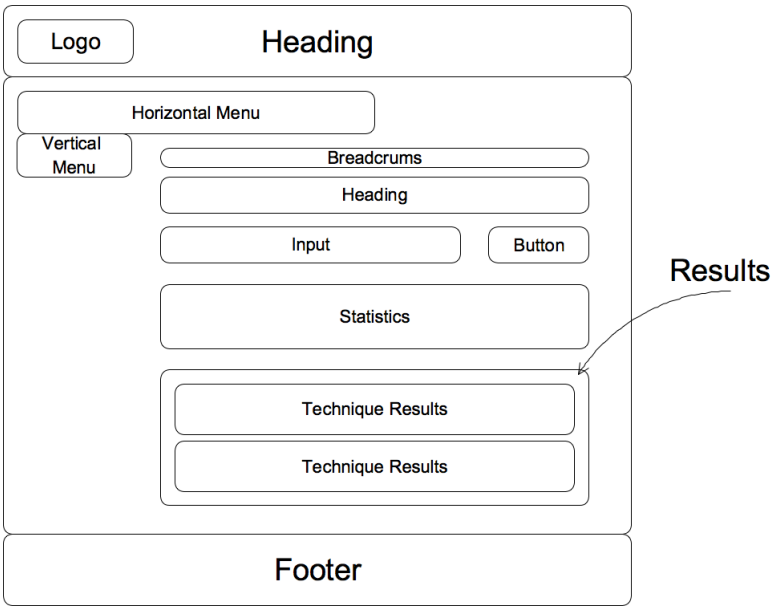

Figure 7.1: QualWeb Interface Prototype

The Statistics area, shows some information regarding the evaluation: how long it took, how many passes, fails and warnings, all three in numbers and percentages. The result area is where the results are presented.

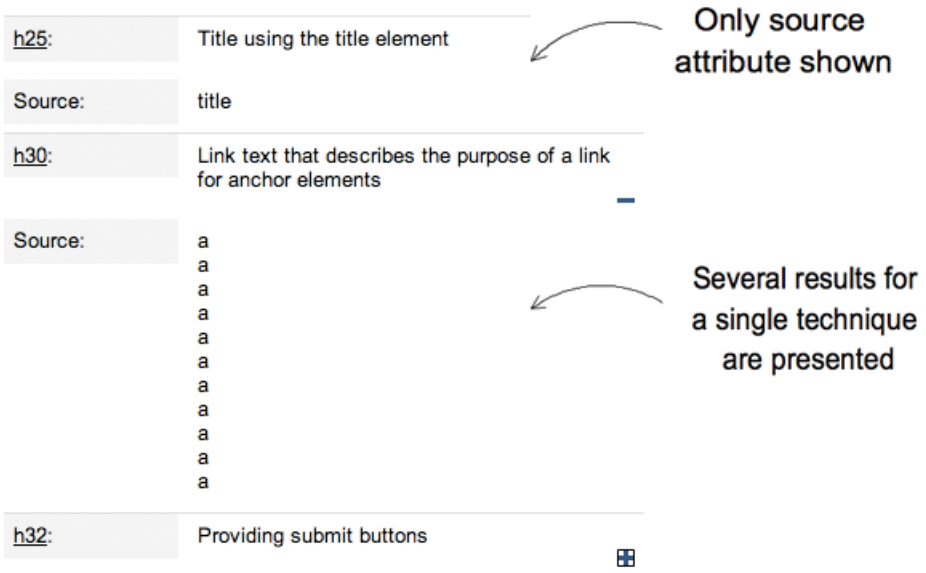

Figure 7.2: QualWeb Interface Results Previously to this work

Figure 7.2 shows how results were presented previously to this work. They appear grouped by technique, so whenever we expanded the technique, all occurrences found, for that technique, were displayed. However, for each one of these occurrences, only the source attribute was printed. This happens in results found for technique H25, indicating that this technique was triggered only once.

Unlike technique H25, technique H30 shows several sources. This means that there were several anchor tags tested in the HTML document, in which the technique found an error.

7.2 Rationale for the new User Interface

This version of the user interface provides a great starting point for the provision of QualWeb. Nevertheless it was not enough to show the full potential of the current version of the tool. Because of this we wanted to strengthen it by adding new functionalities.

7.2.1 Design

For this, we wanted to elevate its Design by:

- Displaying several attributes for each occurrences;
- Making theses occurrences more distinct from each other;
- Keeping the space occupied as compact as possible.

Figure 7.3 is a prototype of what we agreed the display would be like. It shows some extra attributes that help developers locate and understand the accessibility problems:

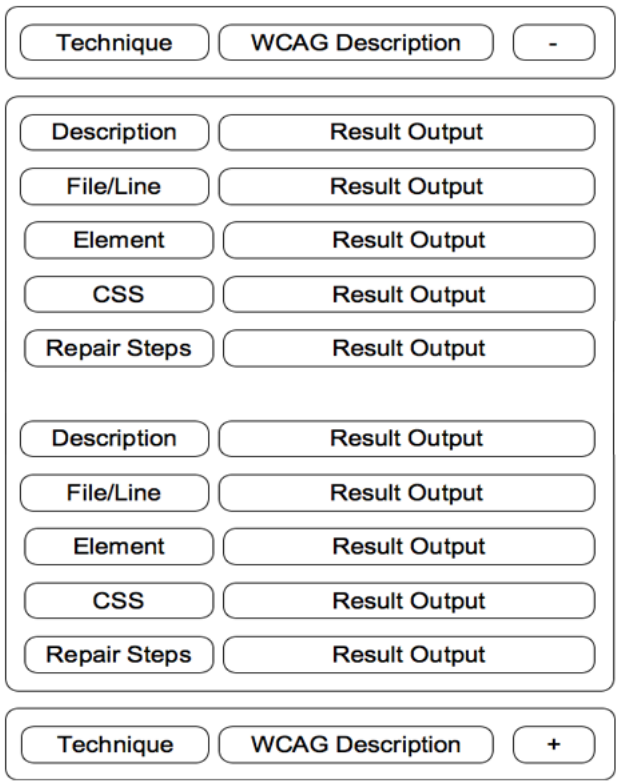

Figure 7.3: Results by technique and attributes prototype

Attributes Specification

Instead of presenting only the ”source” attribute, in the end, for different types of results, different types of attributes are shown:

- For Passes:
  - Description of the Error;
  - File and Line so that the developers can locate the Error in their code;
  - Target Element, the element which the Error refers and its attributes;
- For Fails and Warnings:

- Description of the Error;
- File and Line so that the developers can locate the Error in their code;
- Target Element, the element which the Error refers and its attributes;
- Original CSS containing the excerpt of code where the error was found or the string ”No CSS defined for this element”;
- Repair Steps where we present the repair steps and the notes.

We decided to keep result presentation for passes as compact as possible, avoiding presenting too many attributes. With this in mind we end up only presenting a description and the localization of the element in question. The minimum information in order for the user to know which element passed.

Regarding Fail and Warning results:

- We kept the description, as it was the main way to justify why that occurrence was an accessibility problem.
- We used a single field for File and Line in order to save space in the presentation.
- The element and its attributes, were also presented in a single field, for the same reason. Attributes were not reconstructed inside the element, as we could not be sure that we would be reconstructing them in the same original order.
- Original CSS field keeps the CSS where the problems was found
- Repair Steps field keeps the steps to be followed in order to solve the accessibility problems as well as the Educational Notes, used to provide extra information to the user.

We though that these were the essential attributes to help locate the element. Still, since the number of results is connected to the number of elements in the Webpage, our interface can end up growing considerably. It was for this reason that we tried to join together as many attributes as possible.

7.2.2 Final Result

In this subsection we introduce some details of the final appearance of the results page. Figure 7.4 has a overall view of the results page and figure 7.5 is a detail of that image. Figure 7.5 illustrates the appearance of the evaluation process statistics. For each evaluation it presents the duration of the evaluation, the count of the elements involved in the evaluation and the number of passes, fails and warnings.

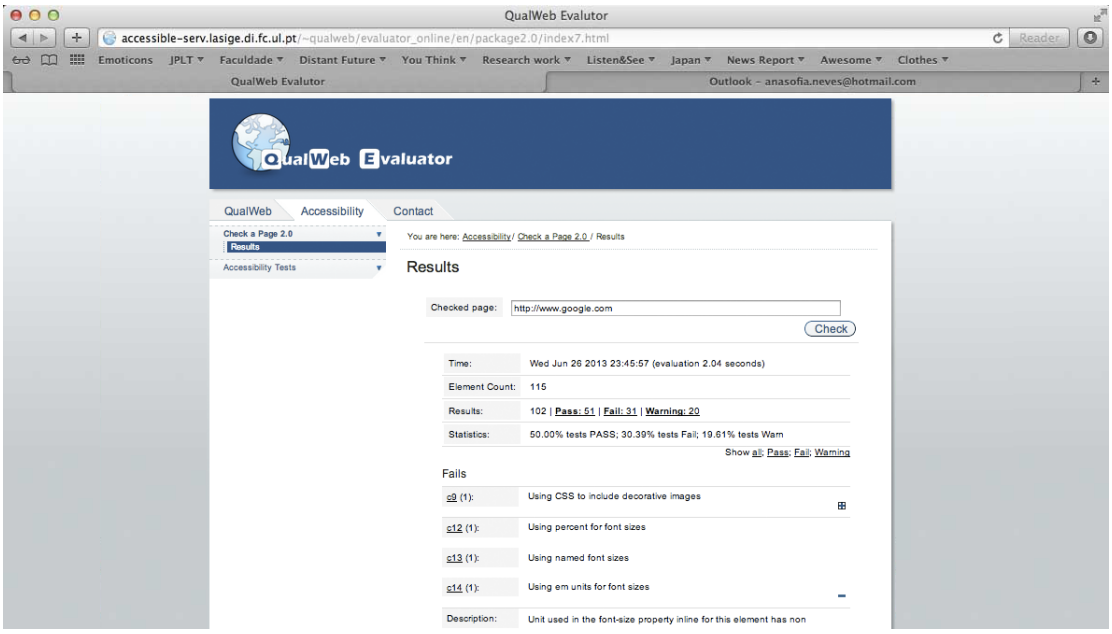

Figure 7.4: QualWeb's results page

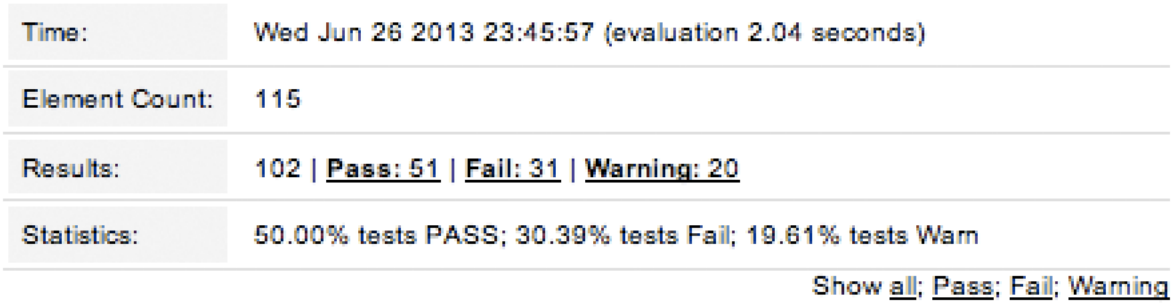

Figure 7.5: QualWeb's results page

Figure 7.6 illustrates another detail of figure 7.4 concerning the results of the evaluation. It shows several techniques, on the left, with the number of occurrences of each one. On the right side we can see a description for the technique, this description is the WCAG 2.0 technique description and not the one we mentioned before.

In this figure we can also see an example of a techniques results expanded. For technique c12 c13 and c14 a single result entry is shown. We can see the fields described before, including the CSS, the repair suggestions and the educational notes.

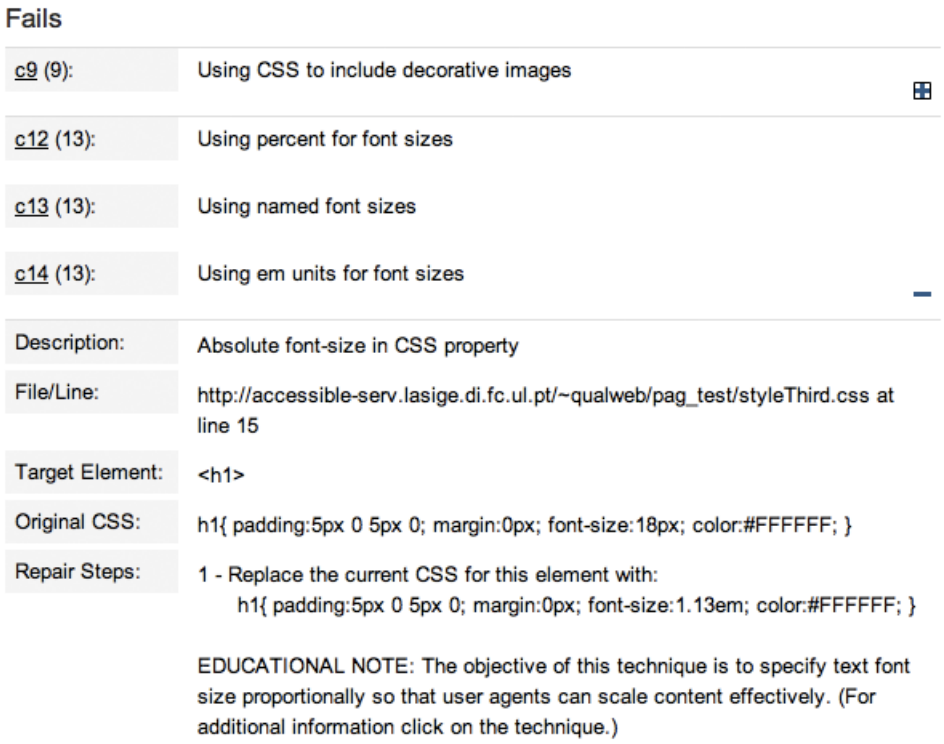

Figure 7.6: QualWeb's fails results presented to the test group

7.3 User Testing

When the development of the interface and the repair tool reached a stable version, we decided to start the user test phase. For this, we created several tasks that, together with a Webpage with some accessibility problems, would be presented to a test group and a control group.

7.3.1 Test scenario

For the user testing we used 20 individuals. 10 in the control group and 10 in the test group. All had at least minimal knowledge of HTML and CSS, although some were experienced developers. Experienced developers were equally spread between the two groups.

All individuals in both groups, were presented the same Web page and were given the same four tasks to complete. The test group used QualWeb user interface with repair suggestions while the control group was presented a version of QualWeb user interface without repair suggestions.

Tests were all conducted in the same computer and in the same Web Browser. Although the computer had a Macintosh operating system, also around half of the individu-

als were Mac users and were spread through both groups. The Browser used was Chrome and all users were experienced users.

Web page edition was done in Dreamweaver, and test individuals experience ranged from, never used, used once or twice, regular users to experienced users. Distribution of these individuals was not registered, however all used the tool without major difficulty.

7.3.2 Test Tasks

Four tasks were created to be presented to the test individuals. Users in both groups had to reach the same solution in order for a task to be complete.

Task 1

On technique c9 locate in the evaluator and repair in dreamweaver the images with

- src=images/logo.gif
- src=images/logoSpacer.gif

Task 2

On technique c12,13,14 locate in the evaluator and repair in dreamweaver the element <h3>

Task 3

On technique c15 locate in the evaluator and repair in dreamweaver the anchor with Original CSS Hover: #menu ul li a:hover.

Task 4

On technique c21

- locate in the evaluator and repair in dreamweaver the element with description Line height with absolute unit
- locate in the evaluator and repair in dreamweaver the element with description CSS in style attribute has line height with value lower than 1.5 `em` or 150 `%`.

7.3.3 Task 1

Task 1 asked users to repair the two decorative images images/logo.gif and images/logoSpacer.gif. The catch here was that only one was a real decorative image, images/logoSpacer.gif, but both showed as accessibility problems for the same reason(Figure 7.7). Because only one of the images was a real decorative image, the user was expected to understand whether the image was a real decorative image or not and for both cases discover how to repair each one.

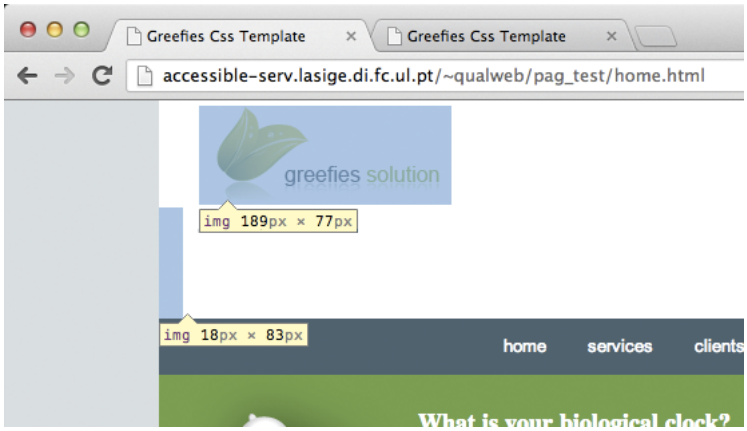

Figure 7.7: Images to be repaired

For the test group things went rather smoothly as they just needed to follow the repair steps they were given, and had no problem with this, although for the control group, this was a little harder to repair. The solution was to add alt text to the logo.gif and to put the logoSpacer.gif as background of a div and formatting it so that it stayed in the same place. But to to this, the user had to understand the concept of a decorative image as well as the accessibility problem they bring.

7.3.4 Task 2

For this task the user was expected to locate the element, to understand the problem and to repair it accordingly. Here the element at fault, <h3 >, has the property font-size with absolute unit. To repair the element, font-size property had to be changed from an absolute unit into a relative unit. The test group was presented with a value conversion whereas the control group had to find out which value to use. Results presented to the control group are shown in figure 7.8.

During the tests, the control group, responding to their own question of what values to use, often used a trial and error approach. When converting to `percentage` users started from low values such as 5% and 20% whilst when converting to `em` units starting

from replacing 12px to 12em, then to 5em, finally reaching the approximate correct value of 1em. Overall `percentage` was the unit users chose more frequently, probably since technique c12 was the first they read.

Figure 7.8: H3 element view for control group

7.3.5 Task 3

In technique c15 the objective was for users to understand that there was a Focus property missing and to understand why. With the test group the Focus property was added, no questions asked, but the control group did not understand what to do. After looking around a little they eventually remembered the technique's link on the top and discover the solution by reading the description of the technique on the W3C webpage(Figure 7.9).

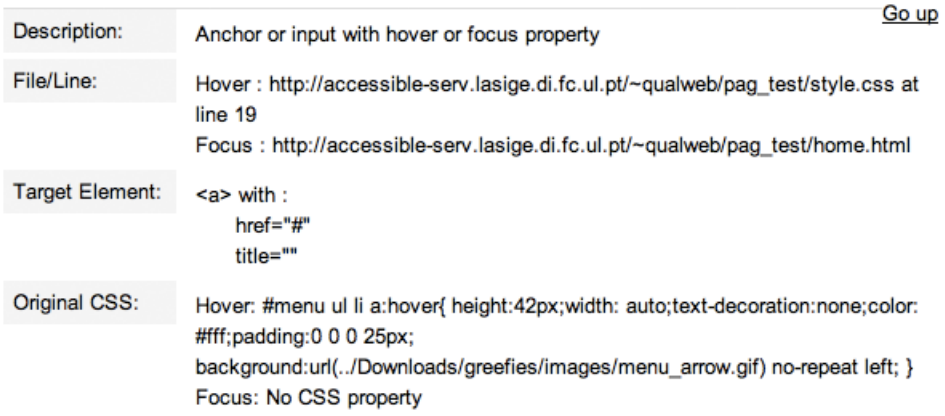

Figure 7.9: c15 result to be repaired

7.3.6 Task 4

For the last technique the user had to locate the first element with absolute unit and the first element where inline CSS had value lower than 1.5 `em`. For the control group, repairing this was apparently easy, they read: "absolute unit" and immediately though: "ah ok, absolute unit, i need to change it to `em` or `percentage`". Although to make them reach

the correct value, always required some coaching. It would be suggested: ”for this one there is a specific value you have to use” and only then, would they reach the correct solution(Figure 7.10).

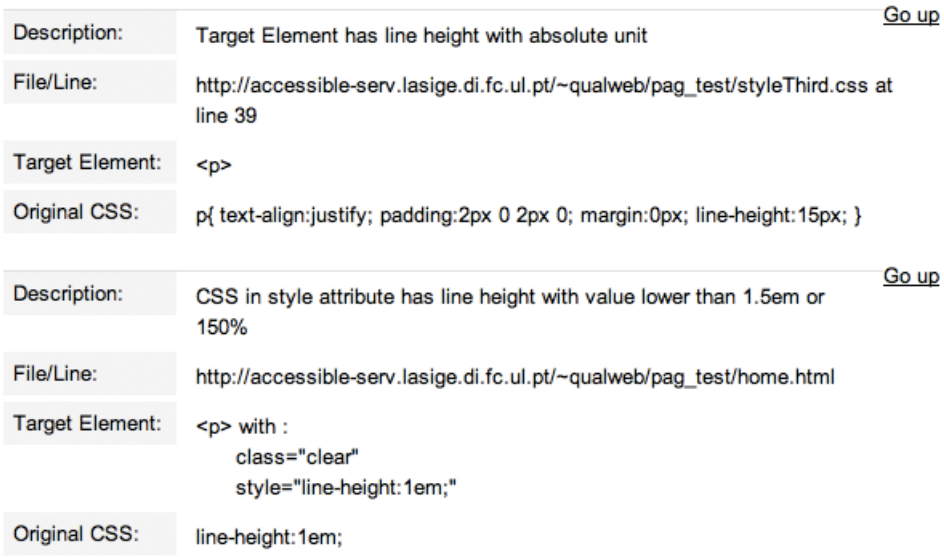

Figure 7.10: c21 result to be repaired

Repairs introduced cause the text to have bigger spacing between lines, this can be seen in figure 7.11.

Figure 7.11: c21 not repaired vs repaired

7.3.7 Test Task Observations

Relating to the evaluation and repair results as well as their presentation, we reached the conclusion that the Webpage would benefit from some changes being applied to its current state. Overall, links to the CSS techniques (the W3C Webpage) needed to be clearer, as most users did not notice them at all, and expand buttons also needed to be a little more noticeable. Although, relating specifically to the results' presentation:

- Descriptions played a bigger role than we gave them credit for. When confused,

users, specially the control group, turned to the descriptions as guidance and sometimes felt confused and were even mislead by them (Figure 7.12).

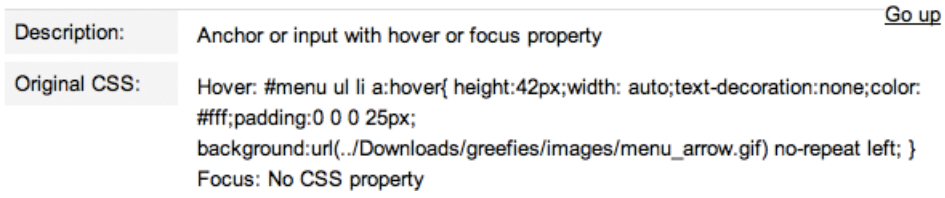

Figure 7.12: Misleading description

- The file/line field was most of the time completely disregarded on the first two tasks as users turned to the control + find shortcut to locate CSS and elements. Our approach to solve this, for the future, is to show a shorten file name instead of a complete URL(Figure7.13), since we believe it's size instead of making it more noticeable achieved the opposite effect.

Figure 7.13: Extended URL file name

Also regarding files, a better solution needs to be applied; as they are now, the file field indicates the HTML file for the error location and when there is a CSS file involved they show the .css file and the line. This situations created a confusing effect when in technique c15 Hover had a CSS property and Focus did not, some users referred: "It says "no CSS property found" but you have the URL up here...". This is visible in figure 6.5.

- Element and its attributes did not raise any questions verbally, but we could still see some struggle to comprehend it, specifically during task 4, where the user was asked to repair the element with inline CSS. Most of the times, user would look and hover the mouse above the element's attributes but then they would identify it as an inline CSS with the following comment: "So, this element is not in a .css file but in the .html file"; which means they located and understood it but not by identifying it in the elements' attributes. Figures 7.14 and 7.15 illustrate the presentation of the elements attributes where the users had to locate the property.

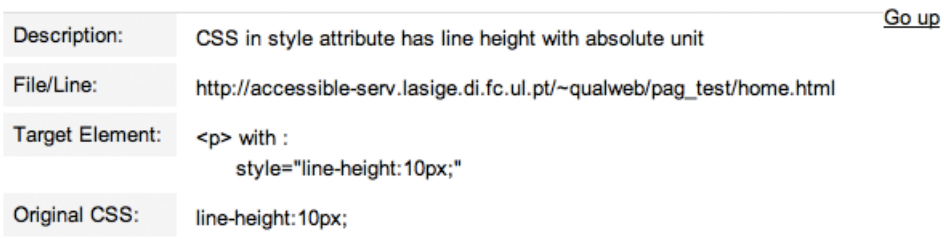

Figure 7.14: Element and attributes in Task 4

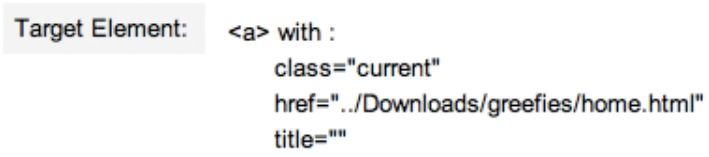

Figure 7.15: Element and attributes other example

- The repair steps, we though they were well accepted and faced as a easy enough solution. Users in the test group would face them with comments such as ”ok” or ”so i just need to copy this into the code?”. We also noticed some difficulty in understanding some more complex steps and these needed to be reformulated.

- Its important to state that the Educational Notes added in the end of the repairs were a success. Users would not only, read it whenever they were not sure of what they were doing (and would immediately understand what they needed to do), but also after they got familiar with the tool, they faced it as a curiosity and said things as: ”Let me just understand why...”.

  The funny thing is that these ”educational notes” (Figure 7.16)consist of nothing more that the first line of the techniques' description in the W3C Webpage, and when the control group had to read these same descriptions, but on the W3C Webpage, they would always read diagonally and most of the times would have to read it twice to understand the same line of text. So in the end this turned out to be a great solution.

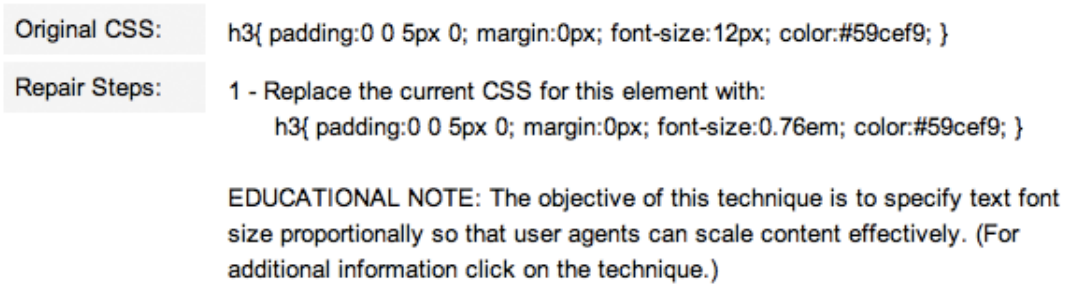

Figure 7.16: Repair steps and educational notes

Relating to the WCAG techniques, we feel it is important to refer that consulting the W3C techniques can not be considered as a normal task that developers will follow. Some of these techniques are not only difficult to interpret, they also use specific accessibility terms. Fair english readers will probably face an added difficulty when reading them and may become unmotivated; also, since we tested our tool with a few HTML newcomers, we can say that some of these techniques are more directed to experienced HTML users, especially people in the accessibility area. For example, none of the 20 people tested knew what a decorative image was in accessibility, or at least in a W3C point of view.

7.4 SUS Results

After the user accomplished all four tasks, a questionnaire was applied. This questionnaire had two parts. The first part consisted of ten questions which compose the system usability scale (SUS) and the second part was a background information questionnaire, which had the purpose of obtaining more information about the user.

Observing figure 7.17 the results obtained from the SUS report applied to the Control Group were:

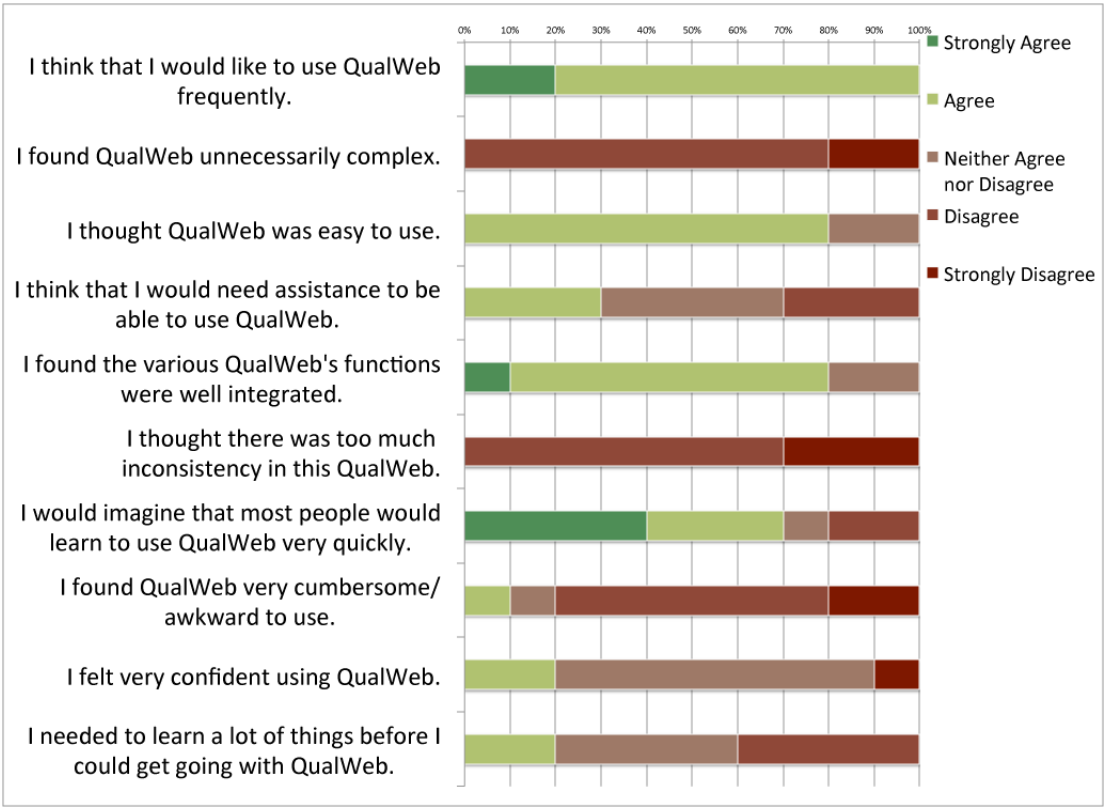

Figure 7.17: User testing SUS Results

- In the fourth bar, 70% of the users were not sure, in the end, if they could use QualWeb on their own. 30% even said they would need help to use QualWeb.
- In the eight bar, 10% confess they felt QualWeb cumbersome to use and another 10% were neutral.
- 70% said in the ninth question that they neither felt confident nor diffident while using the tool and 10% even strongly disagreed to having felt confident while using the tool.

Figure 7.18 show the results for the Test Group, the group who was presented with the repair suggestions:

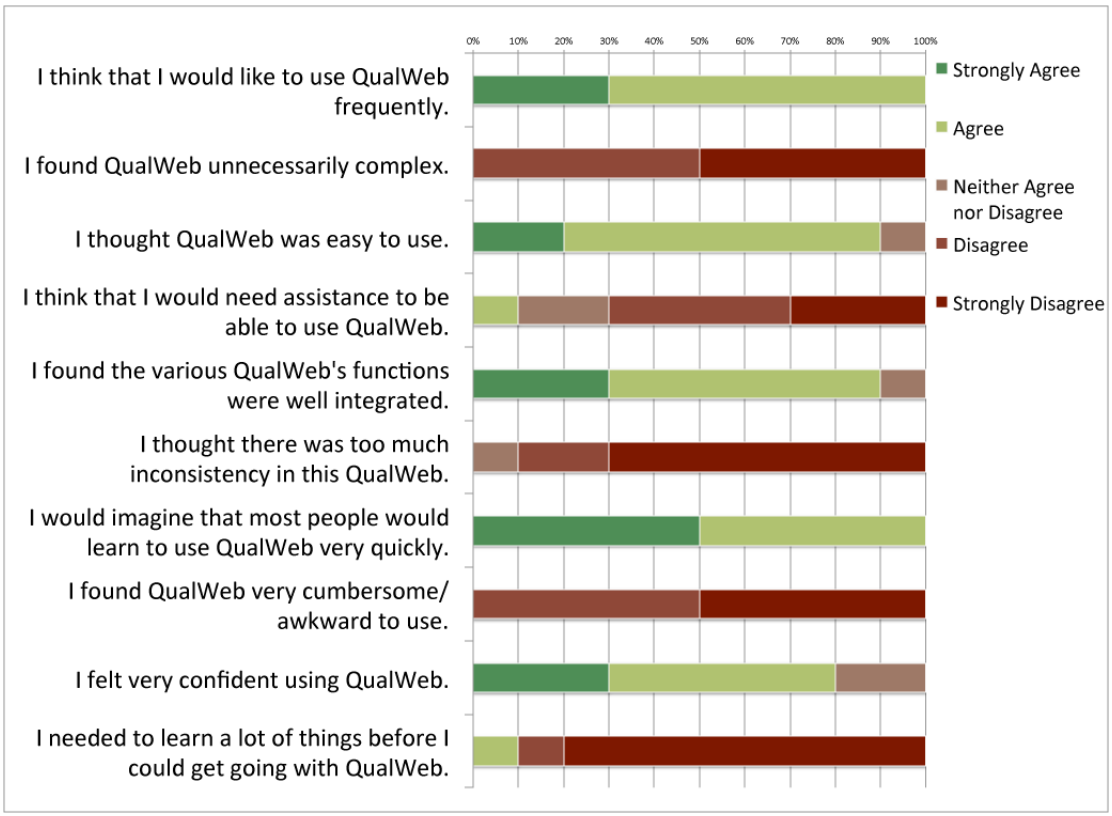

Figure 7.18: User testing SUS Results

From this figure we can make the following observations:

- In bar two, only 10% did not think QualWeb to be easy to use.
- In contrast with the previous image, on bar four, we can see that only 10% said they may need assistance when using the tool, against the 30% in the previous group; and only 20% were neutral.

- Also in contrast, no one felt QualWeb was cumbersome to use (bar eight).
- 20% were neutral when asked if they felt confident while using QW, but the majority of answers tended to the green instead of the red as in the previous image.
- And finally 10% said they would need to learn lots of things but this was a person who also indicated that had only student level HTML and CSS knowledge.

At first glance we can perceive two major things in this new figure. First one is how strong answers are more frequent than in the previous figure, (darker green and red), and the second is in the last bar. The last question asked users if they think they would need to learn a lot of things in order to use QualWeb; we can see that 80% said strongly that they would not need to, in contrast to the 0% from the other group who gave the same answer and the 20% who said they would.

7.5 Other feedback

The second part of the questionnaire gave some more information about the users. We tested twenty people, ten were presented with the repairs, ten were not. These people were all computer science students from our University and had different backgrounds, some knew only student level HTML and CSS, other already developed some personal or commercial Webpages. In each group we had four student level and 6 personal and commercial developers.

Only four people, from the twenty, had already tried to apply some kind of accessibility guidelines, but mentioned only alt text in images and not using plug ins. Most of them did not know there were specific textual techniques with good practices to be followed. Only one from the twenty people with whom we testes QualWeb had ever used a automated evaluation tool and said the following "..It was a bit time consuming, even for a front page of a blog, because some repairs would come in the way of the website functionality, and a new fix would be needed."

Some of the comments gathered from the users in the control group were:

- "I think it is a very useful tool, but it took a bit of time to get used to it."
- "Add a FAQ page or a small tutorial to explain the tool's functions"
- "The reparation process is not always easy to understand"
- "The tool would benefit from showing how to solve the problems"

- ”Links to the techniques are not easily precept”

And comments made from users in the test group:

- ”I was confused by the way the techniques were grouped together when I was about to expand a given set of fail results.”
- ”with practice, using this tool gets easier. Makes repairing simple even for people with few knowledge in Web development”
- ”I liked it. It’s well explained”
- ”Display is good but could be clearer”

These comments were added in this report because they illustrate the differences between the control and the test groups’ experience with QualWeb. It is important to remember that tests with the control group took double the time than the tests with the test group, and were sometimes an hour long.

Overall people liked QualWeb, especially speaking for the test group, they enjoyed using it, were self assured and showed less fatigue in the end of the tests. The control group showed more fatigue and frustration, since they had to read the WCAG techniques and interpreted them themselves.

Comparing tests between groups, we feel that if we are to present repair suggestions, we must make sure they are adequate, or at least, as adequate as we can, and make the notes well visible to the user.

Chapter 8

Conclusions & Future Work

With this work we were able to improve QualWeb in many ways. We increased modularity by reorganizing the evaluation process.The two types of evaluations: HTML and CSS, are more integrated now. Also by changing the tool's approach to the evaluation procedure, QualWeb is now able to display results organized in different ways, instead of just by technique.

We were able to introduce CSS evaluation in QualWeb, by implementing 15 out of 22 (68%) of the CSS WCAG 2.0 techniques. This included also adding a CSS pre-processing module, to retrieve CSS files and internal CSS.

The seed for further suggested repairs has been planted, thanks to: the successful development of CSS repairs, for all the implemented CSS techniques; the adaptation of the return results in the HTML techniques and most of all, by realizing that the 100% automated repair approach is not advisable.

All of these stages were verified, and can still be constantly refined, thanks to the development of coded tests and user tests. In fact, QualWeb now has an enhanced User Interface, with a functionality tested and approved by users.

8.1 Lessons Learned

This work starts with the CSS pre processing and the evaluation. For this we had to gather two types of CSS and preprocess them in order for later matching with the HTML elements. It is important to say that **performance has not been the main focus, from the beginning, and because of this, the evaluation process still lacks optimization**.

Regarding the repair process **our purpose was to build a fully automated repair tool, which would return repaired HTML and CSS files to the developer. However**

this turned out to be the worst way to approach the repair process. Repairing files, without explaining to the user why, without having a notion of the global state of the Webpage and without being 100% sure of what we were writing into the new files, was unreasonable.

The next part focused on adapting QualWeb's interface, so that it would be able to present every information needed by the developer to repair each problem. When this was complete, user tests were conducted and from them we derived that users are indeed receptive to accessibility tools and to follow their recommendations.

We derived that **there is a difference between just showing to the user where the problems are and why, and also showing how to make these errors disappear**. This is shown in figures 7.17 and 7.18, which contain the responses given by the users in a questionnaire after the use of the user interface. These answers show that users feel lest lost, less unmotivated and improve their user experience when they are presented the repairs.

Something else worth mentioning, is that the **probability of the user making erroneous repairs is highly reduced, with the presentation of the repairs suggestions**. For example,in Task 4 referring to line-heights, users in the control group were driven to replace the px unit for a number, according to their will, with a relative unit. This would be ok, only if technique c21 did not restrict the line-heigh property to values between 1.5 em to 2.0 em.

With everything in this report in mind, we can observe how **this work made good improvements in QualWeb's functioning**, and that really makes our tool more interesting and captivating from a users point of view.

As an interesting ending point we have to say that users interviewed, mostly university students, show that, only a few know basic principles of Accessibility and even less know of the existence of these kinds of tools. If this is to be changed, **accessibility concepts should be taught to students as early as possible**. For this, given the feedback obtained during user testing, accessibility tools can play a major role. These tools can help by teaching and raising awareness towards accessibility among developers, from a hands on approach.

Besides, **WCAG 2.0 techniques are not easily understood, even by expert developers**, who cannot be expected to read them extensively, especially during a development process.

8.2 Future Work

Some changes can still be applied to QualWeb, under the scope of this work, in order to improve its functionality, for example:

Some performance issues need to be erased: by introducing hashtables in the evaluation and in the CSS structure and by improving the path list used for obtaining family members of an element.

Problems pointed out, and discovered, during the user testing stage need to be repaired.

Currently line numbers are only shown whenever the error occurs in a CSS file. This needs to be extended also to HTML files. Some users did point this out, by saying it would be useful to have that information in the error report.

As we could see in some figures presented in this report, errors are displayed inside one of three divs one for warnings, one for fails and other for passes. Inside these divs they are listed by technique. Until now QualWeb was only prepared to show report results this way. However recently we started to embrace the idea of organizing the results by WCAG criteria. For this, changes now need to be applied so that the CSS technique's results can start being criteria oriented instead.

Finally, some users mentioned allowing automated repairs, but in away where the developer would have control over which repairs are applied automatically. This was an interesting idea, but further though needs to be given to this, since only few techniques are straightforward enough to be repaired this way.

Appendix A

Diagrams

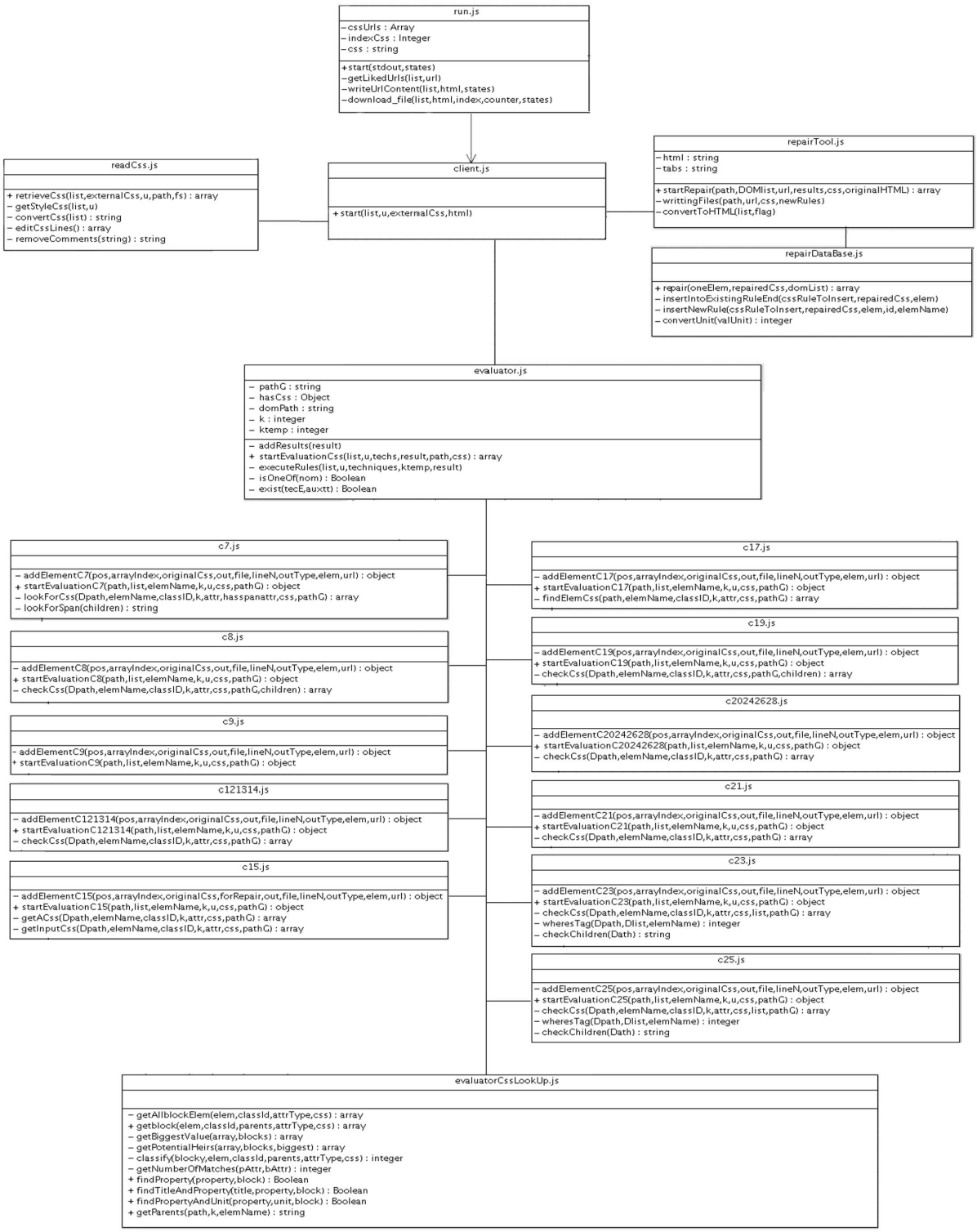

Figure A.1: UML Class Diagram

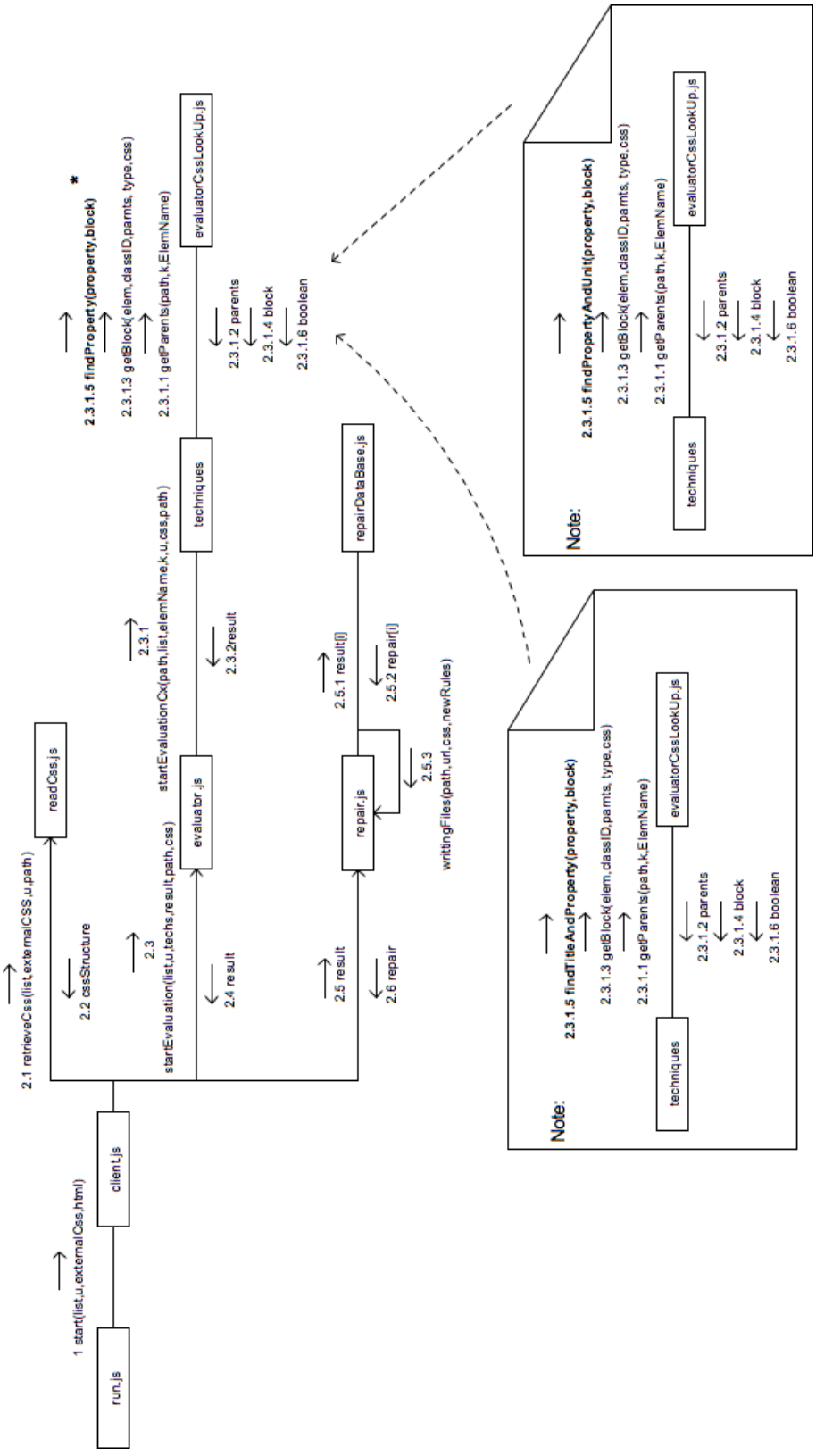

Figure A.2: UML Colaboration Diagram

Appendix B

Papers Written

B.1 Three web accessibility evaluation perspectives for RIA

Three Web Accessibility Evaluation Perspectives for RIA

Nádia Fernandes, Ana Sofia Batista, Daniel Costa, Carlos Duarte, Luís Carriço
LaSIGE/University of Lisbon
Campo Grande, Edifício C6
1749-016 Lisboa, Portugal
{nadiaf,abatista,dancosta,cad,lmc}@di.fc.ul.pt

ABSTRACT

With the increasing popularity of Rich Internet Applications (RIAs), several challenges arise in the area of web accessibility evaluation. A particular set of challenges emerges from RIAs dynamic nature: original static Web specifications can change dramatically before being presented to the end user; a user triggered event may provide complete new content within the same RIA. Whatever the evaluation alternative, the challenges must be met.

We focus on automatic evaluation using the current WGAG standards. That enables us to do extensive evaluations in order to grasp the accessibility state of the web eventually pointing new direction for improvement.

In this paper, we present a comparative study to understand the difference of the accessibility properties of the Web regarding three different evaluation perspectives: 1) before browser processing; 2) after browser processing (dynamic loading); 3) and, also after browser processing, considering the triggering of user interaction events.

The results clearly show that for a RIA the number of accessibility outcomes varies considerably between those tree perspectives. First of all, this variation shows an increase of the number of assessed elements as well as passes, warnings and errors from perspective 1 to 2, due to dynamically loaded code, and from 2 to 3, due to the new pages reached by the interaction events. This shows that evaluating RIAs without considering its dynamic components provides an erroneous perception of its accessibility. Secondly, the relative growth of the number of fails is bigger than the growth of passes. This signifies that considering pages reached by interaction reveals lower quality for RIAs. Finally, a tendency is shown for the RIAs with higher number of states also exposing differences in accessibility quality.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

W4A2013 – Technical, 13-15th May 2013 ˇ Rio de Janeiro, Brazil.

General Terms

Measurement, Human Factors.

Keywords

Web accessibility, Web science, Web browser processing, Automated evaluation.

1. INTRODUCTION

Rich Internet Applications (RIAs) are becoming a new trend on the Web. These are no longer static Web pages but rather complex applications that approximate the behaviour of common native desktop applications. Amongst the vehicles that enable this complexity of RIAs are scripting languages, such as Javascript/AJAX. When executed, these can modify several features, including the page content, layout, etc., within the same URI, sometimes without leaving the browser client side context. As a consequence a URI can correspond to a complex graph of states [19], which are only: 1) complete after all browser processing is done (e.g. load events); 2) reachable though triggering the events (e.g. mouse click, key press) available through each static component (state).

Regarding accessibility, RIAs provide real opportunities and treats. Opportunities, as, for example, they provide means to enrich and diversify the presentation and adapt it to specific users' needs. Enriching the applications may seriously preclude the use of Assistive Technologies (AT) not ready to cope with the dynamic content. Also, a RIA can easily and automatically load hidden components, again and again, containing a significant amount of bad-practices, which are, on first analysis, not detectable.

Evaluating the accessibility in RIAs may, thus, be an exhausting task. From an end-user point of view, each change in presentation and interaction element, reachable through one of the possible RIAs events, should be also checked for accessible compliance. That means letting the browser process the automatic events and following all the interaction opportunities available, covering the whole graph of states of the application. Here, as in large-scale evaluations, automatic evaluators can play a determinant role [14, 17].

Most existing automated evaluators are not capable of assessing dynamically injected content. Also, to our knowledge none is able to fully access the underlying scripting languages. Therefore, the use of a traditional evaluator may lead to an erroneous accessibility evaluation, as several problems and quantities may pass unnoticed. This problem was partially exposed in a preliminary study by *Fernandes et*

al [9]. There, the authors addressed a small number of RIAs and a partial view of the whole page rendering process.

In this paper, we perform a deeper analysis. We extend the sample of assessed Web sites to over 10000, of which more than 8000 included some form dynamic change through user interaction events. Also e use an updated version of *QualWeb Evaluator* [9], implementing a larger thoroughly revised set of Web Content Accessibility Guidelines (WCAG) 2.0 techniques [5]. Finally, we make a comparative analysis of three perspectives of evaluation: 1) corresponding to the traditional approach of evaluation before browser processing; 2) considering the evaluation after browser processing; 3) and, also after browser processing, considering the triggering of user interaction events. In the latter we look into the influence of states in the evaluation outcomes.

2. WEB BROWSING PROCESS

The dynamics of Web pages centre around a sequence of communication steps between the Web browser and Web servers, as depicted in Figure 1.

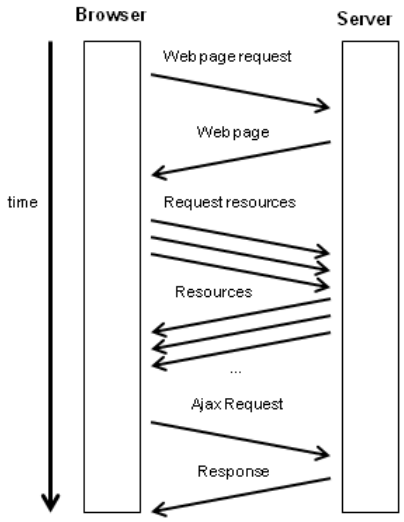

Figure 1: Web Browsing Resource Processing

This communication takes the form of *request-response* interactions, focusing on three main aspects:

- *Web page*: this is the main resource that defines the skeleton of the content that is being presented in the Web browser;
- *Resources*: these complementary resources include images and other media, stylesheets, and scripts that are explicitly specified in the Web page's structure (i.e. with proper HTML elements);
- *AJAX*: these resources are transmitted during or after the browser triggers the loading events for a Web page, allowing server communication and a high level of user interactivity.

This is a mixture between the architecture of the Web (*request-response* nature of *Web pages* and *Resources*) and the Web page loading process within a browser (e.g. *AJAX*).

2.1 Web Page Loading Process

Several steps are executed before users are able to interact with the Web page, as depicted in Figure 2.

The first step in the Web page loading process, a *Request*, concerns with getting the resources that compose the Web page. Another step is the *parsing* of these resources in order to build the HTML Document Object Model (DOM) tree. Afterwards, the Web browser triggers the *DOM Ready* event, when DOM hierarchy has been constructed. Finally, the *DOM Load* event is triggered after all the initial script execution and resources are rendered (e.g. CSS, images, etc.).

Dynamic Web pages often attach a set of behaviours to these events. This way, scripts are executed before the user gets the chance to start interacting. Since the HTML DOM tree is available for manipulation by these scripts, they can enable the addition/removal/transformation of this tree. Consequently, the Web page presented to the user (page after browser processing) might be significantly different from the URI's resource representation that is initially transmitted to the browser from the Web server (page before browser processing).

In Rich Internet Applications (RIAs) new content can be obtained using *Javascript/AJAX*, without refreshing or loading a new page. A user interaction (Figure 2) can easily modify visible elements without requesting data from a server (e.g. through Javascript). Alternatively an AJAX request to the server can fully modify the presented content. In both cases a new version of a page will be available without changing the URI [16, 27], i.e., a new state of the Web page.

Figure 3 shows an example of two possible states of a Web page. Clicking a button ("Button") executes Javascript code that injects more elements in the page with new elements that should be evaluated. Ultimately the AT a user is utilizing must cope with this new content. Note that the URI is the same and the access to the new elements can only be done if the Javascript is interpreted.

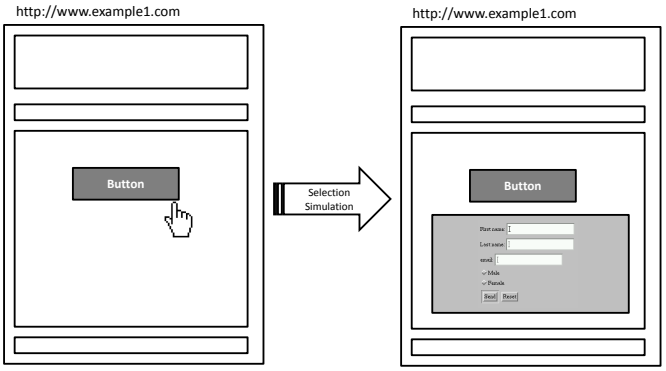

Figure 3: Generate a new state

2.2 Research Hypothesis

Considering the way Web browsers interpret Web pages, as detailed above, and taking into account that users interact with these Web resources through browsers, and possibly ATs, we post that:

When evaluating RIAs, Web accessibility tech-

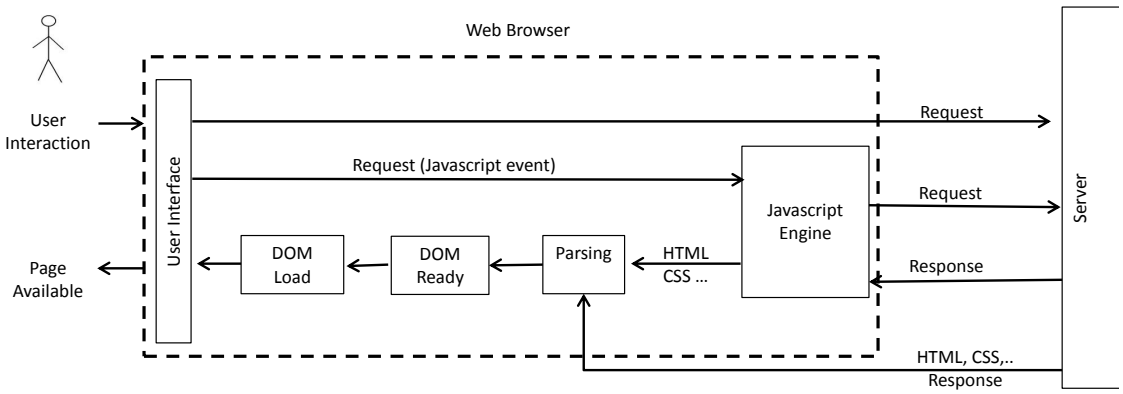

Figure 2: Web Page Loading Process (adapted from [16]).

niques should be applied to what is ultimately presented and interacted with by the end-users, to its full extent (i.e. covering all states).

Of course, alternatives may be envisaged, amongst which, for example, a full source code analysis could be done including scripting interpretation. Also, we recognise that other aspects of RIAs should be taken in consideration, like new complex elements (e.g. canvas) or WAI-ARIA (Web Accessibility Initiative - Accessible Rich Internet Applications) properties [15]. We believe that these should be considering in addition to covering the whole RIA states' graph.

From that rationale, we devised the following research hypothesis that serves as the basis for our experimental study:

> *Evaluating Web content that considers fully processed RIA states will provide different accessibility evaluation outcomes.*

3. RELATED WORK

Web Accessibility Evaluation (WAE) is a procedure to analyse how well the Web can be used by people with different levels of disabilities [14]. Unfortunately, current studies show that many Web sites still cannot be accessed in the same conditions, by a large number of people [14, 17].

WAE can be performed in two ways: users' based or experts' based. The users' based evaluation is carried-out by real users; they can verify how well the accessibility solutions present in the Web sites match their own needs. However, assessment by users is often subjective. Furthermore, user testing is necessarily limited in scale, thus leaving a substantial number of problems out.

Experts' based evaluation can be performed manually or automatically. The first is focused on the manual inspection of Web pages. Contrarily to the one above, it is performed by experts and it can provide a more in-depth answer to the accessibility quality of a Web page. This type of evaluation should be considered as a complementary evaluation rather than a replacement of user-evaluation. However, it is a time-consuming process too, and it can bring potential bias in the comparison of the different Web pages' accessibility quality [14, 17].

The automatic evaluation is performed by software. The expertise is embedded in a software framework/tool. The evaluation is carried out by the tool without the need of direct human intervention. The main benefits are scalability and objectivity [17], enabling the conduction of large-scale studies, like [18] or [8] for example. However, it has limitations that both direct or user's evaluations do not have, e.g., the depth and completeness of analysis. Again, it is a trade-off and often constitutes a complement to manual evaluations.

Experts' evaluations rely on knowledge. Especially for the automatic version, the focus of this work, that knowledge is expressed in a set of guidelines, preferably in a way that can be automated.

Web Content Accessibility Guidelines (WCAG) [3] are one of the most used technical standards for accessibility evaluations, encouraging creators (e.g., designers, developers) in constructing Web pages according to a set of best practices. If this happens, a good level of accessibility can be guaranteed [14, 17]. These guidelines can be used in automatic evaluations.

The results of an accessibility evaluation can be used to measure quantitatively the level of accessibility of a Web page. Metrics are important to facilitate understanding, control, and improvement of products and processes in software development [11]. In terms of accessibility, metrics can also help the user to understand if a Web page/site can be used by them. In addition, they provide an important tool to understand whether companies may improve the accessibility of final products or companies beginning the software development can introduce accessibility issues in the development process.

The metrics' results can be obtained with the aid from automatic evaluations. Some examples of metrics include: Failure Rate [21], UWEM [22], and WAQM [23]. Some authors say that the metrics should not be dissociated from the users, and consider the necessary effort to perform the repair of the accessibility problems of the pages [6, 20]. We concur. However, metrics can also be used to understand the Web accessibility behaviour through large-scale evaluation with hundreds or thousands of evaluations. That does not mean dissociating metrics from users. It simply means that some users view them macroscopically.

3.1 In Browser Evaluation

The predominant technologies in the Web were HTML

and CSS, which resulted in static Web pages. However, current Web pages, by means of user or automated events, can change their content. Thus, the final presented content can be different from the initially loaded by the Web browser [10].

Unfortunately, most of the current automated evaluators [1] are still not capable of detecting those changes. Nevertheless, some can be pointed that already work on after browser processed DOM trees, or in Browser Context. Examples are *Foxability*, *Mozilla/Firefox Accessibility Extension* and *WAVE Firefox toolbar* [12]). Unfortunately they are still not using WCAG 2.0. Another example is *HeraFFX 2.0* [13]. Being a semi-automatic evaluator, the evaluation outcome can be improved and more accurate through human intervention. However, this approach is hardly compatible with large scale evaluations. In general, most of the existing tools aim at interactive evaluation, on a page by page, or small scale basis.

Table 1 presents several Web accessibility evaluators currently used, considering: 1) if they perform the evaluation before processing, 2)the version of WCAG used, 3)if they perform the EARL reporting (standard format for Web accessibility report), and 4)some notes that we considered appropriate. Through the analysis of this table, we can conclude that none of these evaluators assess RIAs, nor meet the requirements for the characteristics we have identified as relevant.

In RIAs this problem is aggravated, because of the states that can be triggered in the same page. Current evaluators would not be able to identify this changes and consequently not able to evaluate their accessibility problems. Therefore, these technologies are used and combined in new ways that threatens accessibility [4].

Besides, the increasing usage of video components on the web is becoming a relevant problem when considering, for instance, deaf or blind people. Text alternatives should be made available. Dynamically created content and *AJAX* based Web pages are growing and most of them are not considered accessible as screen readers, such as *JAWS*, do not seem to work satisfactorily [26]. Dynamic Web enables the change of the Web page's content and structure, usually by displaying or hiding information and HTML elements, injecting new HTML code or even removing it [10]. As can be seen, guaranteeing that the advantages of this new *"Web of applications"* are available to everyone, demands for a complete and rigorous accessibility evaluation process capable of handling the characteristics of these type of Web pages.

3.2 Rich Internet Application coverage

A fundamental problem in evaluating RIAs' accessibility is identifying the complete states-graph of the application. *Watanabe et al* [24] tested the accessibility requirements in RIAs. Their proposal simulates keyboard events to change the DOM tree of the Web pages. The objective was to produce AT scenarios and to know if it was possible to navigate through the page with those events. However, their system did not allow them to simulate all the possible user interactions with the page.

Mesbah et al [19] presents a tool to perform automated tests of AJAX applications, taking into account the dynamics of these applications. The strategy was to access event changes by adopting a Web crawler capable of triggering the events on, for example, the *clickable* elements of the user interface. Still, the tool confines its crawling to AJAX requests and does not provide accessibility evaluation means.

Recently, *Doush et al* [7] designed a conceptual RIA accessibility evaluation tool. The idea of this framework is to check the accessibility of the visible content of a Web page. Also it provides a report with the content that should appear in the Web page (considering ARIA specifications). However, this framework is not yet implemented.

4. QUALWEB

To perform the accessibility evaluation, we used the QualWeb evaluation framework [9]. The architecture (depicted in Figure 4) is composed by three major modules: QualWeb evaluator core, Browser Processing Simulator, and Interaction Simulator.

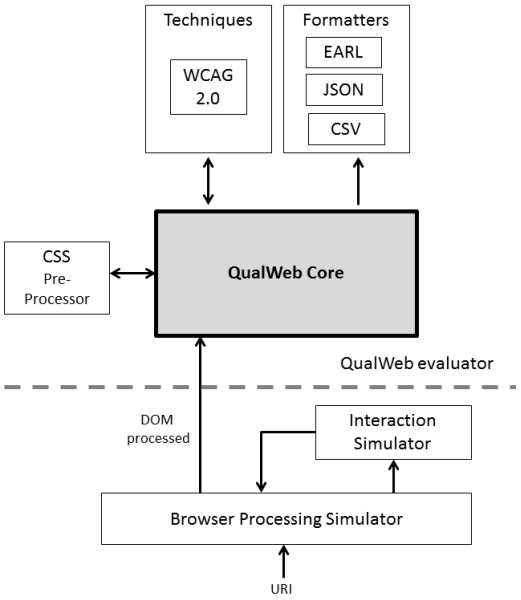

Figure 4: Architecture of QualWeb.

4.1 QualWeb core

The Browser Processing Simulator (BPS) receives the URI of the page to evaluate and process it. If the page has states, it sends the DOM of the page to the Interaction Simulator, which simulates the several states. Then, the Interaction Simulator send them to the BPS to be processed, which posteriorly forwards the processed DOM of the pages to the QualWeb core to be evaluated.

This way, the DOMs are then fed to the QualWeb evaluator core that cumulatively assesses the quality of the RIA.

To extract the CSS from the DOM tree is used the CSS Pre-Processor which obtains all the CSS of the page (i.e., internal, external and in-line). Next, to perform the evaluation *QualWeb* uses the features provided by the *Techniques* component. It uses the *Formatters* component to tailor the results into specific serialisation formats, such as: EARL reporting [2], since EARL is a standard format for accessibility reporting; comma-separated-values (CSV) for statistical proposes; or JSON if we want to use QualWeb as a Web service. If the results report are formatted in EARL, it can be interpreted by any tool that understands this format, and

Table 1: Web Accessibility evaluation tools. BBP - evaluation before Browser Processing, ABP - evaluation after Browser Processing.

Name	BBP	ABP	WCAG 1.0	WCAG 2.0	RIA	EARL Reports	Notes
A-Checker	√			√			
TAW Standalone	√		√			√	
Foxability		√	√				
Functional Accessibility Evaluator		√					In upgrade to WCAG 2.0. It used alternative guidelines to WCAG.
WAVE		√	√				
Hera-ffx 2.0		√		√		√	Semi-automatic tool

even allow comparing the results with other tools.

In this version of QualWeb, we added more techniques. We currently implement 31 HTML and 13 CSS WCAG 2.0 techniques (i.e., the basic practices of creation of accessible content as basis of testable success criteria). Thus, a total of 44 accessibility techniques were used for the evaluation.

4.2 Browser Processing Simulator

This module simulates the processing of the Web browser using *PhantomJS*[1]. *PhantomJS* is a command-line tool that uses WebKit. WebKit is an open source web browser engine used in some of the most disseminated web browsers [25]. *PhantomJS* works like a WebKit-based web browser, interpreting web applications, without rendering them on a display a headless WebKit browser with a JavaScript API.

The absence of display rendering, still interpreting the applications, enables this module to perform sequential evaluations in browser context, in a fast and thought-out way. This favours large-scale evaluations. The access to after the Web browser processing is achieved through the *onLoadFinished* event available in the *PhantomJS* API.

4.3 Interaction Simulator

To cope with the challenges of the RIAs, we have integrated an Interaction Simulator. This component is responsible for simulating user actions and triggering the interactive elements of the interface. As a result we have access to all the different states (in DOM format) of RIAs.

To perform the simulation, we use JQuery to interact with *clickable* elements (e.g.,*button*, *input*, *div*). For every state we get the children states (those reachable through clickable elements) and processed them in the BPS. Then, for each child we verify if it was already visited. If not we repeat the process. At the end we obtain the full interaction state graph of the RIA. Figures 5 and 6 show the DOM trees of the initial and state resulting of click in the link.

4.4 Validation and Testing

4.4.1 WCAG Techniques

A test-bed was developed, in order to verify that all the WCAG 2.0 implemented techniques provide the expected results. The test-bed is constituted by a set of HTML and CSS test documents, based on documented WCAG 2.0 techniques and ancillary WCAG 2.0 documents. Besides, each HTML test document was carefully hand-crafted and peer-reviewed (within the research team), in order to guarantee a high level of confidence on the truthfulness of implementation. Success or failure cases were performed for each technique, to test the possible techniques' outcomes (unit tests). First static examples were implemented, e.g. static html code that produces fail/warn/pass according to the guidelines. Then, the test-bed also considers dynamic cases. For each technique a corresponding test that dynamically generates the same code was created. That code can be generated on the page load or as a result of a user interaction. The evaluation outcomes (warn/pass/fail by technique) for all HTML/CSS test documents were compared and checked for consistency.

Additionally, we performed an expert analysis and compared its results with the ones of QualWeb. We performed it on some of the pages the more used Web sites (from Alexa Top 100 Web sites[2]), using WCAG 2.0 also. We inspected before browser processing, after browser processing, and after processing considering the states of the pages. For the implemented techniques the results were consistent, with the exception of the warnings. Those were considered fails by the experts in some cases and passes in others.

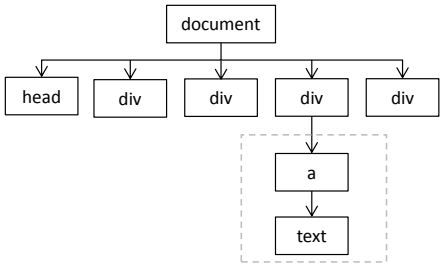

Figure 5: DOM tree of a initial state of a page.

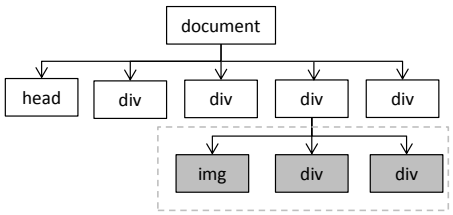

Figure 6: DOM tree of the second state of a page.

[1] http://phamtomjs.org/

[2] http://www.alexa.com/topsites

4.4.2 States detection

To verify the validity of the Interaction Simulator we performed unitary tests, using the dynamic tests of the testbed. That allowed us to verify the coverage as well as the adequacy of the algorithm. To finalize the validation of the states, we also compare the Interaction Simulator with the tool from *Mesbah et all* [19][3]. Again we verified that all the *clickable* events, triggered by *Mesbah et all* tool, were equally triggered by our Interation Simulator.

5. EXPERIMENTAL STUDY

Three outcomes are gathered:

We used the QualWeb to compare Web accessibility evaluations under the following three conditions:

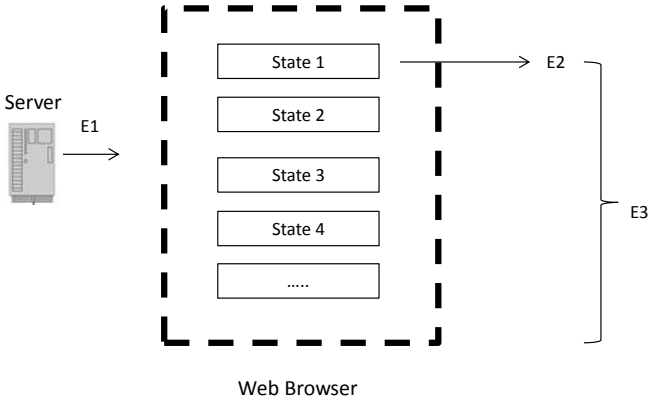

Figure 7: Representation of types of evaluation.

- **Evaluation 1 - E1:** evaluation performed before Web browser processing – Interaction Simulator and Browser Processing Simulator turned off.
- **Evaluation 2 - E2:** evaluation performed after browser processing, without consider states of the pages –Interaction Simulator turned off and Browser Processing Simulator active.
- **Evaluation 3 - E3:** evaluation performed after browser processing, considering the different states of the pages – both modules active.

Figure 7 present a schematic representation of the evaluations.

5.1 Setup and Measurement

We gathered 14000 URI by crawling the Web, in middle of January 2013. The seed was a Portuguese portal, although the obtained URI were not confined to the Portuguese domain. The list of URI was split and feed into 20 instances of the QualWeb evaluator running in a corresponding number of PCs. The evaluation took approximately 10 hours.

The results of the evaluation are presented in terms of *PASS, WARN and FAIL*: *pass* or *fail*, if the elements (or certain values/characteristics of the elements) verified by the techniques are in agreement or disagreement with the W3C recommendations for the techniques, respectively; and *Warning* – if it is not possible to identify certain values/characteristic of an element as right or wrong, according to the W3C recommendations for the techniques (without a need of an expert intervention).

We used *strict rate* metric [18], where WARN results are not considered ($strict\ rate = pass/(pass + fail)$). It is normalized into a percentage, where the results are between *accessible* (100%) and *not accessible* (0%). This is of course not an absolute value of accessibility. Amongst others, warnings should be disambiguated and the techniques should be integrated in success criteria, for example. However, since we are comparing the results in the different conditions, the comparison between this metric's results provides a sufficient indication of the relative accessibility quality.

5.2 Results

Our evaluations observed a total of 11860 viable URIs, and 20869 CSS files. A total of 2140 URIs were no longer available. From these pages, we withdrew the ones with only one state. This way we focused in understanding the states influence on the evaluation results. Finally, we proceed with a total of 8282 URIs.

The average number of evaluated elements per RIA was: 1152.21 elements in E1; 1665,7 elements in E2; and 19964,00 elements in E3 (Figure 8). The number of elements in E2 is higher than in E1 ($ratio_{E2/E1} = 1.45$), and the number of elements is higher in E3 then in E2 ($ratio_{E3/E2} = 11.99$).

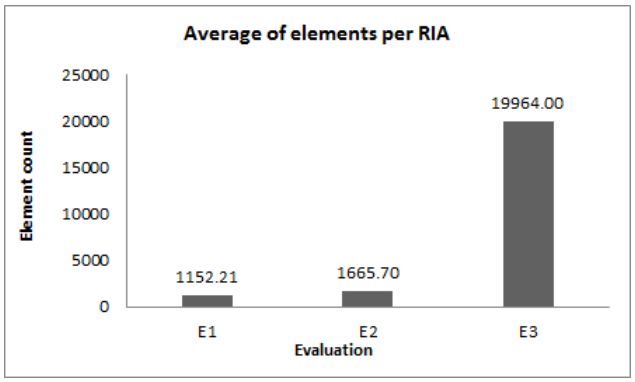

Figure 8: Average of number of HTML elements by type of evaluation.

Considering the states ratio in E3 we get 1715.10 elements per RIA, per state: a number similar to the average number of elements in E2. This means that the states reached by the interaction events have a similar number of states that the main one. A closer look shows a standard deviation of 4.13.

5.2.1 Evaluation Outcomes

Figure 9 presents the outcomes of the evaluations. It can be observed that the outcomes increase in E2 relatively to E1 and in E3 relatively to E2. The average numbers of outcomes per RIA are relevant for the analysis as they show what has been ignored if states are not considered.

We also obtained the average number of outcomes per RIA, per state, to understand if states behave similarly to the main URI. The figure 10 shows these values. In this case the average numbers are similar. We notice a slight decrease

[3]http://crawljax.com/

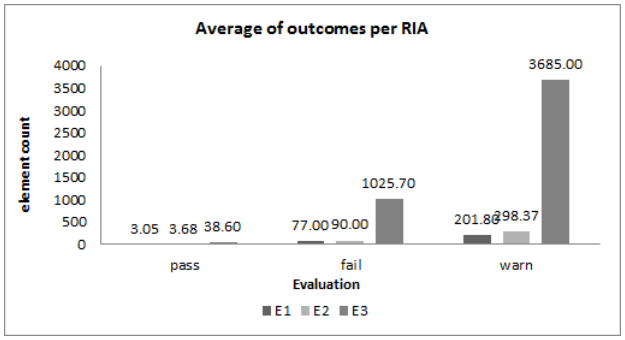

Figure 9: Comparing average outcomes per RIA.

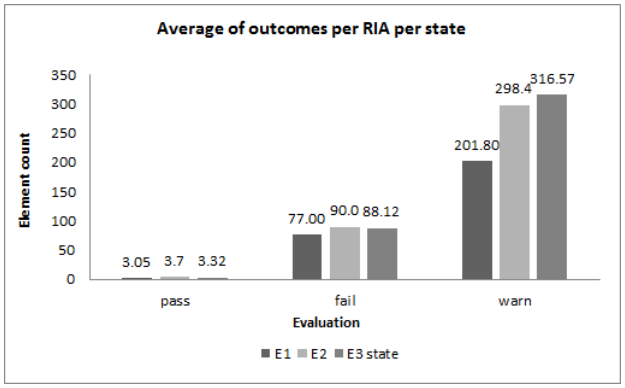

Figure 10: Comparing average outcomes per RIA, per state.

on the number of passes and fails from E2 to E3 per state, and a growth of warnings.

Table 2 show the ratios from the evaluations, including the E3 per state values. Most values show an increase from E1 to E2 and E2 to E3, especially in the latter. Pertaining to the values per state, the ratios of E3 per state and E2 shown a slight decrease in the number passes and fails. Nevertheless, it is worth noting that the fails ration is bigger than the pass.

Table 2: Ratios per outcome.

Ratio	Pass	Fail	Warn
E2/E1	1.21	1.17	1.48
E3/E2	10.49	11.40	12.35
E3 states/E2	0.90	0.98	1.06

Considering states, we detected on average 12.51 possible states per URI, in E3. This means that, on average, each URI can have approximately 12 different states, triggered by users' interaction that will be presented to the AT they use. The average of the standard deviations of the outcomes per RIA shows that the accessibility differences between main URI and the remainder assessed states in small: average pass per RIA per state is 3.32 (SD 0.02); average fail per RIA per state is 88.12 (SD 0,30); average warning per RIA per state is 316,57 (SD 0,65).

Regarding the use of the strict metric, we have found a small decrease in accessibility quality from E2 to E3. However, in both cases the quality is really low (0.041 for E2 and 0.039 for E3) taking into to account that 0 (zero) is the lower value (no passes) and 1 is the higher value (no fails) of accessibility quality.

Figure 11 shows the results of the strict metric for RIAs with increasing number of states. A trend line was included revealing a tendency on accessibility quality with the increasing number of states. We excluded the number of states

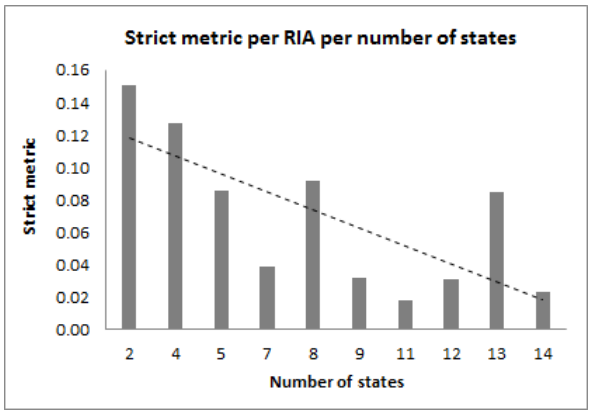

Figure 11: Comparing the results of the strict metric on RIAs, considering the number of states.

that had less that 5 represented RIAs. For example, there was a single RIA with 16 states and two with 25 that were therefore excluded from the figure. An interesting characteristic of the URI with 16 states (www.radiosim.sapo.pt) was that the strict metric result was a fairly high comparing with all the others (0.17). A closer look revealed a radio web site.

6. DISCUSSION

Our study has yielded interesting insights respecting the automated Web accessibility evaluation practices. Revisiting our initial research hypothesis we in fact can confirm that:

> *Evaluating Web content that considers fully processed RIA states will provide different accessibility evaluation outcomes*

According to the posted rational that Web accessibility techniques should be applied to what is ultimately presented and interacted with by the end-users, then we may conclude that accessibility evaluation should be applied to whole set of RIAs states.

The results obtained show that E3 presents a higher average of outcomes, relatively with E2 and E1. The case for E1 and E2 was discussed elsewhere [10]. The case for E3 derives directly from the number of evaluated elements as each interaction provides a new state.

A closer look revealed that in a majority of RIAs the new states are comparable, but not equal, to the main URI. Interaction merely changes some data on the destination state or a small number of attributes. As a consequence the difference of the evaluation outcome are also small. Also shown by the small standard deviation.

Although this is true, however, the errors introduced by these changes would not be detected by a classic automatic

evaluation method. That means that a user may find a barrier on an otherwise barrier-free RIA, just because he/she interacted with it. Dynamic content has shown by these results, and those discussed in [10], does in fact hinder accessibility barriers.

Another interesting aspect of the above results emerges from the relation between the number of states of a RIA and the perceived accessibility quality. These may reveal that increasingly complex RIAs are prone to have more accessibility problems. This conclusion needs to be further investigated.

The cost/benefit has also to be considered. For instance, E3 outcomes do provide a more exhaustive accessibility evaluation of RIAs, and in the end it will provide a more detailed view of the RIA accessibility issues. Nevertheless, for a large number of states (and of pages) it can be overwhelming.

Designers can develop more accessible content, if they use evaluation procedures that consider all the possible states of the Web application. This happens because they have access to a more complete page/applications analysis, which they may use to improve the accessibility quality of the page/application.

6.1 Limitations

Our experiment has faced some limitations on the type of results that can be extrapolated, including:

- *Techniques coverage:* it would be important to have ARIA techniques implemented and adopt new emerging techniques that will, for sure, appear from the conformance to HTML 5 new features;
- *States flow-graph:* the states detection algorithm has to be improved so that it becomes able to detect the complete flow-graph of the RIA. For now, we were more focused in *clickable* events, but events such as *onFocus* may, for instance, as well originate changes in the content of the page;
- *Duration of evaluation:* to minimize the duration, instead of evaluating a new state as a new page when we find it, we should ponder to compare its DOM with the original page DOM . If this operation takes less time than evaluating all the new states of the page, we should only evaluate the new elements. This is particularly important in large-scale evaluations;
- *Cross-Site scripting:* in some Web pages the injection scripts are blocked with *cross-site scripting* (XSS) dismissal techniques. In these cases, we are not able to inject JQuery (if necessary) to simulate the interaction with the pages;
- *Automated evaluation:* since this experiment is centred on automated evaluation, it shares all of the inherent pitfalls.

Next we present the conclusions of the experiment, and try to synthesize the important points.

7. CONCLUSIONS AND FUTURE WORK

This article presented a large-scale study of accessibility on over 8000 RIAs. The results of this study allowed us to compare three accessibility evaluation approaches: before and after Web browser processing, and after browser processing with states.

Web pages have become more complex and evolved from simple information display into RIAs. We can conclude that there are, in fact, differences between these three approaches. Thus, we were able to verify that Web Accessibility Evaluations after browser processing, considering all the possible states of the page, really provide a more accurate and in-depth analysis of page accessibility.

In conclusion, regular Web accessibility evaluations or even evaluators which consider browser processing but without states, overlook 92% of the states of the pages (for 11860 Web pages). Consequently, the accessibility problems on those states are ignored. This is a reality that has to be changed, so that Web pages accessibility may be improved.

As for the future, we aiming to improve the states detection algorithm; and detect the actual differences between states of a RIA. With that we can not only find the impact of states for end user accessibility, but also have a measure of the effort needed to correct the page.

8. ACKNOWLEDGEMENTS

This work was funded by Fundação para a Ciência e Tecnologia (FCT) through the QualWeb national research project PTDC/EIA-EIA/105079/2008 and through the FCT national research projec PTDC/EIA-EIA/117058/2010.

9. REFERENCES

[1] S. Abou-Zahra. Complete List of Web Accessibility Evaluation Tools, march 2006. Available from: http://www.w3.org/WAI/ER/tools/complete.

[2] S. Abou-Zahra and M. Squillace. Evaluation and report language (EARL) 1.0 schema. Last call WD, W3C, Oct. 2009. Available from: http://www.w3.org/TR/2009/WD-EARL10-Schema-20091029/.

[3] L. R. G. V. B. Caldwell, M. Cooper. Web Content Accessibility Guidelines 2.0. W3C Note, World Wide Web Consortium (W3C), December 2008. from http://www.w3.org/TR/WCAG20/.

[4] M. Cooper. Accessibility of emerging rich web technologies: web 2.0 and the semantic web. In *Proceedings of the 2007 international cross-disciplinary conference on Web accessibility (W4A)*, W4A '07, pages 93–98, New York, NY, USA, 2007. ACM.

[5] M. Cooper, L. G. Reid, G. Vanderheiden, and B. Caldwell. Techniques for WCAG 2.0 - Techniques and Failures for Web Content Accessibility Guidelines 2.0. W3C Note, World Wide Web Consortium (W3C), October 2010. Last accessed on November 26th, 2010, from `http://www.w3.org/TR/WCAG-TECHS/`.

[6] M. Cooper, D. Sloan, B. Kelly, and S. Lewthwaite. A challenge to web accessibility metrics and guidelines: putting people and processes first. In *Proceedings of the International Cross-Disciplinary Conference on Web Accessibility*, W4A '12, pages 20:1–20:4, New York, NY, USA, 2012. ACM.

[7] I. A. Doush, F. Alkhateeb, E. A. Maghayreh, and M. A. Al-Betar. The design of ria accessibility evaluation tool. *Advances in Engineering Software*, 57(0):1 – 7, 2013.

[8] N. Fernandes and L. Carriço. A macroscopic web accessibility evaluation at different processing phases.

In *Proceedings of the International Cross-Disciplinary Conference on Web Accessibility*, W4A '12, pages 18:1–18:4, New York, NY, USA, 2012. ACM.

[9] N. Fernandes, D. Costa, S. Neves, C. Duarte, and L. Carriço. Evaluating the accessibility of rich internet applications. In *Proceedings of the International Cross-Disciplinary Conference on Web Accessibility*, W4A '12, pages 13:1–13:4, New York, NY, USA, 2012. ACM.

[10] N. Fernandes, R. Lopes, and L. Carriço. Evaluating web accessibility at different processing phases. *New Review of Hypermedia and Multimedia*, 18(3):159–181, 2012.

[11] A. P. Freire, R. P. M. Fortes, M. A. S. Turine, and D. M. B. Paiva. An evaluation of web accessibility metrics based on their attributes. In *Proceedings of the 26th annual ACM international conference on Design of communication*, SIGDOC '08, pages 73–80, New York, NY, USA, 2008. ACM.

[12] J. L. Fuertes, R. González, E. Gutiérrez, and L. Martínez. Hera-ffx: a firefox add-on for semi-automatic web accessibility evaluation. In *W4A '09: Proceedings of the 2009 International Cross-Disciplinary Conference on Web Accessibililty (W4A)*, pages 26–34, New York, NY, USA, 2009. ACM.

[13] J. L. Fuertes, R. González, E. Gutiérrez, and L. Martínez. Developing hera-ffx for wcag 2.0. In *W4A '11: Proceedings of the 2011 International Cross-Disciplinary Conference on Web Accessibililty (W4A)*, New York, NY, USA, 2011. ACM.

[14] S. Harper and Y. Yesilada. *Web Accessibility*. Springer, London, United Kingdom, 2008.

[15] S. L. Henry. WAI-ARIA Overview. W3C Recommendation, World Wide Web Consortium (W3C), January 2011. Available from: http://www.w3.org/WAI/intro/aria.

[16] W. Kern. Web 2.0 – End of Accessibility? Analysis of Most Common Problems with Web 2.0 Based Applications Regarding Web Accessibility. *International Journal of Public Information Systems*, 4(2):131–154, 2008.

[17] R. Lopes and L. Carrico. Macroscopic characterisations of web accessibility. volume 16, pages 221–243, Bristol, PA, USA, Dec. 2010. Taylor & Francis, Inc.

[18] R. Lopes, K. V. Isacker, and L. Carriç. Redefining assumptions: accessibility and its stakeholders. In *Proceedings of the 12th international conference on Computers helping people with special needs: Part I*, ICCHP'10, pages 561–568, Berlin, Heidelberg, 2010. Springer-Verlag.

[19] A. Mesbah and A. van Deursen. Invariant-based automatic testing of ajax user interfaces. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 210–220, Washington, DC, USA, 2009. IEEE Computer Society.

[20] S. Mirri, P. Salomoni, L. A. Muratori, and M. Battistelli. Getting one voice: tuning up experts' assessment in measuring accessibility. In *Proceedings of the International Cross-Disciplinary Conference on Web Accessibility*, W4A '12, pages 16:1–16:4, New York, NY, USA, 2012. ACM.

[21] T. Sullivan and R. Matson. Barriers to use: usability and content accessibility on the web's most popular sites. In *CUU '00: Proceedings on the 2000 conference on Universal Usability*, pages 139–144, New York, NY, USA, 2000. ACM.

[22] E. Velleman, C. Meerveld, C. Strobbe, J. Koch, C. A. Velasco, M. Snaprud, and A. Nietzio. Unified Web Evaluation Methodology (UWEM 1.2), 2007. Available from: http://www.wabcluster.org/.

[23] M. Vigo, M. Arrue, G. Brajnik, R. Lomuscio, and J. Abascal. Quantitative metrics for measuring web accessibility. In *W4A '07: Proceedings of the 2007 international cross-disciplinary conference on Web accessibility (W4A)*, pages 99–107, New York, NY, USA, 2007. ACM.

[24] W. M. Watanabe, R. P. M. Fortes, and A. L. Dias. Using acceptance tests to validate accessibility requirements in ria. In *Proceedings of the International Cross-Disciplinary Conference on Web Accessibility*, W4A '12, pages 15:1–15:10, New York, NY, USA, 2012. ACM.

[25] Webkit. The webkit open source project, 2011. Available from: http://www.webkit.org/.

[26] M. Zajicek. Web 2.0: hype or happiness? In *Proceedings of the 2007 international cross-disciplinary conference on Web accessibility (W4A)*, W4A '07, pages 35–39, New York, NY, USA, 2007. ACM.

[27] X. Zhang, Y. Zhang, and J. Wu. Research and analysis of ajax technology effect on information system operating efficiency. In L. D. Xu, A. M. Tjoa, and S. S. Chaudhry, editors, *Research and Practical Issues of Enterprise Information Systems II, Volume 1, IFIP TC 8 WG 8.9 International Conference on Research and Practical Issues of Enterprise Information Systems (CONFENIS 2007), October 14-16, 2007, Beijing, China*, volume 254 of *IFIP*, pages 641–649. Springer, 2007.

B.2 Web Accessibility in Africa: a Study of Three African Domains

Web Accessibility in Africa: a Study of Three African Domains

Daniel Costa*, Nádia Fernandes*, Sofia Neves*, Carlos Duarte*, Raquel Hijón-Neira**, Luís Carriço*

*LaSIGE, University of Lisbon, Portugal / **Universidad Rey Juan Carlos, Spain

{dancosta, nadiaf, abatista, lmc, cad}@di.fc.ul.pt,
raquelhijon@gmail.com

Abstract. Being the most used method for dissemination of information, especially for public services, it is of paramount importance that the Web is made accessible as to allow all its users to access the content of its pages.
In this paper, we evaluated 2250 Governmental Web pages from each one of three different African countries (i.e., Angola, Mozambique and South Africa). This report compares the accessibility quality and the level of structural complexity of these African countries government's Web pages. We found that hand coded pages tend to have larger number of HTML elements and also to present higher number of accessibility problems. Finally, it suggests some recommendations to repair the most common problems in these pages.

Keywords: Web Science, Web accessibility, automated evaluation.

1 Introduction

In many countries, the Web is the main vehicle used by governments to spread information, education, allow civic participation and other public services. It also is an important medium for receiving and providing information and interacting with society. Therefore, it is essential that the Web is accessible in order to provide equal access and equal opportunity to people with or without disabilities. Besides, an accessible Web has the potential to help people with disabilities and the elderly to participate more actively in society.

The United Nations (UN) estimates that approximately 10% of the world's population are persons with disabilities [2]. It is difficult to estimate how many people are affected by Web accessibility problems, nevertheless, if we move forward to an ideal situation, where only a reduced percentage of the population faces accessibility barriers, then technology is serving society in the right way.

The importance of Web accessibility is increasing in the international context, and especially in the European Union [1]. In Europe, more and more countries have legislation requiring that government Web sites be accessible. In contrast, developing countries in Africa have less stringent laws, if any [2]. Governments worldwide have several

stimuli to adopt accessibility. Demonstration of social responsibility by provisioning information and services to all citizens is one of them.

In this paper, we present a report of the state of Web Accessibility in three countries located in the African continent. The evaluation of accessibility we describe is based on the Web Accessibility Guidelines (WCAG) 2.0 [3].

1.1 Web Content Accessibility Guidelines 2.0

To help creating accessible Web pages, WCAG 2.0 defines guidelines that encourage designers/developers to craft Web pages according to a set of best practices. These guidelines are also used for accessibility evaluation.

WCAG 2.0 contains several guidelines written as testable sentences and chosen to address specific problems related with accessibility. Each guideline has a testable success criterion, which is supported by techniques that can be true or false when testing Web content against them.

Although, it is possible to use the guidelines to manually evaluate Web pages, due to the nature of this study (i.e., the large number of Web pages evaluated) we used an automated evaluation tool: QualWeb [4].

1.2 QualWeb

QualWeb is a Web automatic accessibility evaluation tool. The main advantage of this tool is the in browser context evaluation [6], i.e., after the Web browser processes the Web page and all resources are loaded. To this end, the Webkit-based Phantom[1] headless browser is used, allowing us to assess the page's code after browser processing. In terms of techniques, QualWeb covers 51% of the HTML and 73% of the CSS techniques.

An additional distinguishing feature of this tool is the ability to find different states of the Web page [4]. This means QualWeb is capable of interacting with DOM elements and detecting changes to the DOM of a page. QualWeb stores a new state if more than content is replaced after interaction (e.g., introduction of new HTML elements on the DOM tree). We consider the total number of states found, the level of complexity of a Web page as this reflects the dynamism we can find on the current state of the Web.

2 Experimental Study

For this study, the first step was to obtain a list of governmental Web pages for each of the three countries: Angola, Mozambique and South Africa. Starting from each of the main government's pages, we used a Crawler to look for clickable elements in it. Every time a clickable element redirected to another URL on the same domain name (gov.ao for Angola; gov.mz for Mozambique; or gov.za for South Africa), this new URL was kept as an object to be evaluated and the algorithm continued to execute.

[1] PhantomJS: http://phantomjs.org/

Using this method, we collected a sample of 2250 government Web pages, from each country.

Afterwards we performed the evaluation itself, on each one of these 2250 Web pages per country. Every Web page was assessed with the QualWeb evaluator to check for conformance with WCAG 2.0 HTML and CSS techniques. The evaluation produced a list of Warnings, Passes and Fails that are analysed in the results section. In the interest of classifying the complexity of the evaluated Web pages, the QualWeb feature allowing the identification of different page states was used to determine the total number of states in the pages evaluated.

2.1 Results

Our evaluation yielded differences in the HTML documents in terms of number of HTML elements, between domains of different countries (Figure 1). The pages of South Africa (za) present a higher number of elements with an average of 846.37 elements per page, followed by pages of Angola (ao) with an average of 360.17 elements per page and finally by the pages of Mozambique (mz) with an average of 344.60 elements per page.

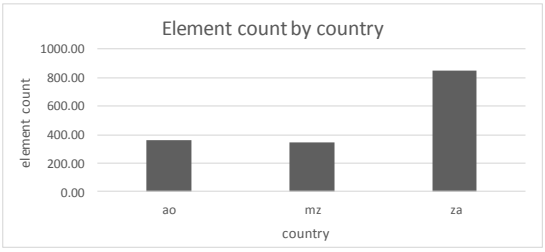

Fig. 1. Average number of elements per page for each country's governmental pages.

Figure 2 presents how the evaluation outcomes (fail, pass and warning) differ between the African countries' Web pages. A *failure* occurs in the cases where the evaluator can detect automatically and unambiguously if a given HTML element has an accessibility problem. A *pass* ensues from elements that, unambiguously, are classified as having no accessibility problems. *Warnings* are raised when the evaluator can partially detect accessibility problems, but which might require additional inspection (often by experts). Table 1 presents the percentage of outcomes (pass, fail and warning) by country. Inspecting these results with additional detail, the Web pages have the following evaluation outcomes:

- **Fail**: Even though the compliance with accessibility techniques is quite different in all three countries, the common factor between the Web pages of Mozambique and South Africa is that fails are slightly above 50%. In addition, the Angolan Web pages are just above 40% for fails.
- **Pass**: Angola's governmental Web pages register the highest percentage of passing elements, reaching over 40%. Mozambique ratio decreases to around 37% and South Africa registers the lowest value, around 19%.

- **Warning**: Mozambique's Web pages elements register the lower percentage of warnings, around 10%. Followed by Angola's Web pages with 13% and South Africa with 27%. The three countries have total of fail and warning above 50%: Mozambique just above 60%; Angola around 55%; and South Africa approximately 80%. South Africa registers the highest total of potential accessibility problems.

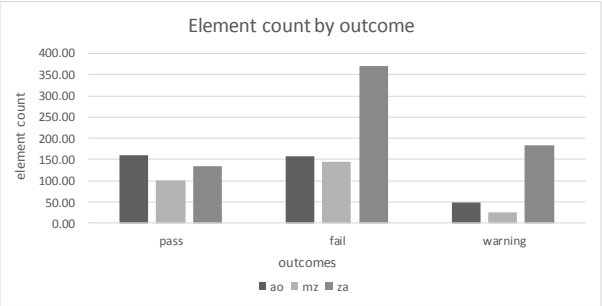

Fig. 2. Average number of HTML elements by evaluation outcome by country.

Table 1. Distribution of evaluation outcomes (absolute values and percentages) by country.

Country	Pass	% Pass	Fail	% Fail	Warning	% Warning
Angola	159.66	**43.61%**	157.92	**43.14%**	48.51	**13.25%**
Mozambique	101.45	**37.42%**	143.77	**53.03%**	25.88	**9.55%**
South Africa	133.47	**19.46%**	370.05	**53.94%**	182.45	**26.60%**

Evaluation by technique

In the following analysis, we will focus on the accessibility results by technique, identifying the more compliant and the more infringed techniques for each country. Figure 3 shows the techniques where occurred passes and their average. All three countries present higher pass values for techniques C23 and C19. The third higher pass value is C8 for South Africa, C9 for Mozambique and C21 for Angola. These techniques evaluate the following conditions:

- C23 – if div elements in main content have background colour;
- C19 – whether text is incorrectly altered to "look" as if it has an align right or centre;
- C8 – for paragraphs and headings, looks for a wrong usage of extra spaces between letters to simulate the letter spacing property;
- C9 – whereas decorative images are specified in CSS rules and therefore removable when disabling CSS;
- C21 – checks if the line-height property is used with relative values and if these values range between the ones recommended.

The first observation that can be made is that HTML techniques present lower values of pass comparatively with CSS techniques. This can be explained by the fact that CSS

techniques are more specific than HTML ones, which means that an automated evaluation can more easily determine pass for these, while HTML return higher number of warnings.

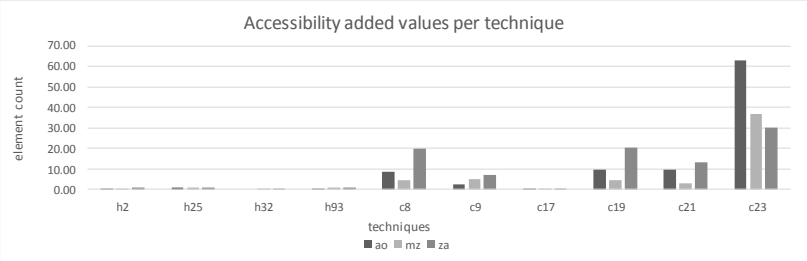

Fig. 3. Average number of passes by technique per country.

The average number of possible problems and problems (fails and warnings) per technique is presented in Figure 4. All three countries present higher values in techniques C15 and C7. For South Africa and Angola, the subsequent high value technique is H30, while for Mozambique is H73. Respectively these techniques evaluate the following conditions:

- C15 – if anchor and input form components present a visual alteration when interacted with;
- C7 – whether anchor elements are followed by a span tag with a textual description of the link hidden by a CSS rule;
- H30 – if the link text describes the purpose of the anchor;
- H73 – checks the correct usage of the summary attribute in tables.

From these results, we can deduce the most common elements with potential accessibility problems. In South Africa and Angola these are anchors or input form components, and in Mozambique tables are added to these.

Incompliance with certain techniques is more pronounced in some countries. For instance:

- H33 – if a title attribute supplements a link, is a more common problem in South Africa (average of 24.61), comparing with the other countries (average of 2.95 for Angola and 1.50 for Mozambique);
- C23 – which presents an average of 9 elements with problems for Angola, being negligible in the other two countries;
- H39 – verifies the usage of caption elements to associate data tables captions with data tables, shows the same behaviour as H33, with an average of 39.58 for South Africa (average of 1.10 for Angola and 11.42 for Mozambique).

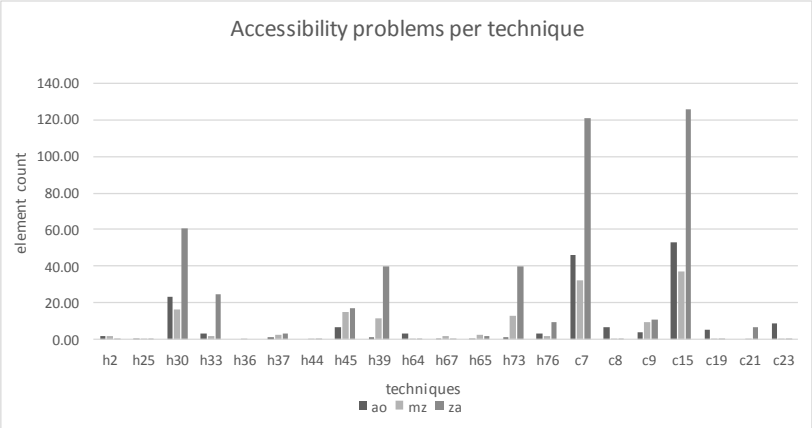

Fig. 4. Average number of fails and warnings (possible problems) by affected technique.

Level of Complexity

We found that the average complexity for the three different domains is approximately 1. The results gathered for Mozambique and South Africa show that the highest level of complexity is 2 (found in 2 Web pages). For the Angolan Government pages, the highest level is 3 (found in 3 Web pages), while 17 pages had level 2.

From these results we can conclude that, for these countries, dynamic changes to the governmental pages layout or interaction elements (thus excluding changes to their content) is not common. When these changes are required, a new page will be loaded, instead of changing the DOM.

3 Discussion

We found there are differences between the three African domains government's Web pages accessibility quality. The South African Government's Web pages have a larger number of HTML elements, but also present a larger percentage of elements raising fails and warnings, comparatively with the Web pages from Angola and Mozambique. This goes towards the conclusions of *Lopes et al* [5], where it was found that the size of the pages influences its quality (i.e., smaller Web pages have less accessibility problems than bigger ones).

Concerning the techniques, it was observed that CSS techniques have a greater influence on the positive accessibility values for all the countries domains than HTML techniques. Techniques C23, C21, C19, and C8 were found to be the ones with highest compliance levels.

When considering potential accessibility problems (fails and warnings), we perceived that they also have higher values in CSS techniques but the difference to HTML techniques is not as pronounced as we found when analysing passes. The techniques most often violated were C7, C15, H30 and H73. It is interesting to note that what we

observed for one of the techniques with more problems, H30 (which verifies if the link text describes the purpose of the anchor), is consistent with what was already seen in a previous accessibility study of two hundred of the most used Web pages in the entire world [6].

The majority of the HTML problems found are related with the accessibility quality of tables, specifically when they do not have captions and summary elements, and if links do not have text descriptions. If those were carefully reviewed and redone the accessibility quality of the pages would considerably improve.

The results show that government Web pages would greatly benefit from reviewing their CSS, since the majority of their problems are located in techniques C7 and C15, especially for the South African government's Web pages. Problems with these techniques can be solved by adding a description of the link given in the anchor element, inside a span tag and hidden by a span CSS, as recommended by the WCAG 2.0 description. For technique C15, the solution would be to ensure that every anchor link and input box changes its colour whenever it is interacted with. People would greatly benefit from this visual aid and contrary to technique C7, it is much easier to enforce. Correcting these situations would help separate the normal paragraph's text and the interactive text in the anchor element, as well as help signalling which form input element is selected at a specific instant when it is being interacted with.

After finishing the automated evaluation, we performed a manual inspection of some of the government's pages from each country. This inspection was performed following the indications of the WCAG 2.0. For the South Africa's Web pages we observed that: the limitations of the several divisions of the pages was not always clear; link elements were confused with parts of the text; the general structure was quite similar to a newspaper and did not denote a lot of accessibility concerns. For Mozambique's Web pages, decorative images do not have either alt or title attributes when they should have them with empty values; some colours are also perceived as being too bright; table captions were also almost inexistent; there are also some flash objects directly embedded without any textual descriptions. For Angolan Web pages, since they generally follow the same structure, they all could benefit from adding captions to tables and textual descriptions to images and anchor elements. We can see that some of the issues found manually confirm the findings of the automated evaluation.

It was also possible to detect that Angola and Mozambique's Web pages benefited from tools that help code generation (such as Flyout and Plone, respectively). On the other hand, South African pages, taking into account the quantity of comments in the code and its specificity, were probably manually coded. This probably contributed to the bigger number of CSS problems, because code generators avoid several CSS problems, such as the use of relative font-sizes.

Regarding the level of complexity of the Web pages, we found that dynamic changes to the pages' DOM are mainly used to change the content of the pages and not to add new elements to the page (i.e., less structural complexity). In what concerns the accessibility quality, the slightly higher complexity found in Angolan Web pages does not reflect any significant change in the overall accessibility score.

4 Conclusion

The Web is the main vehicle used by many governments to spread information, education, allow civic participation and other public services. If these pages are not accessible they fail to reach their target population.

In this paper we evaluated 2250 Governmental Web pages from each one of three different African countries: Angola, Mozambique and South Africa. This report shows that the South Africa Government Web pages have more elements than the other countries but have less quality in terms of accessibility. The Angolan Government Web pages scores the best ratio of passes when comparing with the other countries. Mozambique's pages have the lower rating of fails and warnings combined. Regarding the level of structural complexity, we did not find major differences between the different countries' Web pages.

A manual inspection of a sample of the pages suggested that Angolan and Mozambican Web pages might have benefited from the support of code generation tools during their development, while this is not so clear in South African Web pages. The accessibility evaluation, concomitantly, has shown more accessibility problems in South African pages, with some of these problems being in some cases more easily addressed and prevented with the use of code generation tools.

This overall view of the current state of accessibility in these African governments Web pages by WCAG 2.0 techniques facilitates establishing a set of recommendations to repair the most common problems.

References

1. eAccessibility – Opening up the Information Society. European Commission, December 2010, last accessed on March 20th 2013, from http://ec.europa.eu/information_society/activities/einclusion/policy/accessibility/index_en.htm/.
2. Kuzma, J., Yen, D., Oestreicher, K.: Global e-government Web Accessibility: An Empirical Examination of EU, Asian and African Sites. In: Second International Conference on Information and Communication Technologies and Accessibility, Hammamet, Tunisia, pp.83-90 (2009)
3. M. Cooper, L. G. Reid, G. Vanderheiden, and B. Caldwell. Techniques for WCAG 2.0 – Techniques and Failures for Web Content Accessibility Guidelines 2.0. W3C Note, World Wide Web Consortium (W3C), October 2010. http://www.w3.org/TR/WCAG-TECHS/.
4. Nádia Fernandes, Daniel Costa, Sergio Neves, Carlos Duarte, and Luís Carriço. 2012. Evaluating the accessibility of rich internet applications. In *Proceedings of the International Cross-Disciplinary Conference on Web Accessibility* (W4A '12). ACM, New York, NY, USA, Article 13, 4 pages. DOI=10.1145/2207016.2207019
5. Rui Lopes and Luis Carriço. 2010. Macroscopic characterisations of Web accessibility. *New Rev. Hypermedia Multimedia* 16, 3 (December 2010), 221-243.
6. Nádia Fernandes, Rui Lopes, and Luís Carriço; Evaluating Web Accessibility at different processing phases; New Review of Hypermedia and Multimedia, 2012

Bibliography

[1] Kevin L. Crow Four Types of Disabilities: Their Impact on Online Learning. TechTrends January 2008

[2] Stephanie Hackett and Bambang Parmanto and Xiaoming Zeng Accessibility of Internet Websites through Time ASSETS04, October 2004

[3] W3 Introduction to HTML 4.0. http://www.w3.org/TR/REC-html40-971218/intro/intro.html

[4] Melissa Dawe Desperately seeking simplicity: how young adults with cognitive disabilities and their families adopt assistive technologies CHI '06

[5] Loretta Guarino Reid and Andi Snow-Weaver WCAG 2.0:A Web Accessibility Standard for the Evolving Web W4A '08

[6] Sheryl Burgstahler, Tracy Jirikowic, Beth Kolko and Matt Eliot Software Accessibility, Usability Testing and Individuals with Disabilities EASI Dec 2004

[7] Shari Trewin and Brian Cragun and Cal Swart and Jonathan Brezin and John Richards Accessibility Challenges and Tool Features: An IBM Web Developer Perspective W4A '10

[8] Jonathan Lazar and Alfreda Dudley-Sponaugle and Kisha-Dawn Greenidge Improving web accessibility: a study of webmaster perceptions. 2003 Elsevier Ltd.

[9] Nádia Fernandes, Ana Sofia Batista, Daniel Costa, Carlos Duarte, Luís Carriço Three Web Accessibility Evaluation Perspectives for RIA International Journal of Public Information Systems, 4(2):131 154, 2008.

[10] Walter Kern Web 2.0 End of Accessibility? Analysis of Most Common Problems with Web 2.0 Based Applications Regarding Web Accessibility. ICWE 2009 San Sebastián, Spain, June 24-26, 2009 Proceedings

[11] David Flanagan JavaScript: The Definitive Guide: Activate Your Web Pages 2011

[12] Amaia Aizpurua, Myriam Arrue, Markel Vigo and Julio Abascal Exploring Automatic CSS Accessibility Evaluation ICWE 2009 San Sebastián, Spain, June 24-26, 2009 Proceedings

[13] Greg Gay and Cindy Qi Li AChecker: Open, Interactive, Customizable, Web Accessibility Checking. W4A '10

[14] Kentarou Fukuda and Shin Saito and Hironobu Takagi and Chieko Asakawa Proposing New Metrics to Evaluate Web Usability for the Blind. CHI '05

[15] Hironobu Takagi, Chieko Asakawa, Kentarou Fukuda and Junji Maeda Accessibility Designer: Visualizing Usability for the Blind. ASSETS'04

[16] Leonard R. Kasday, Ph.D. A Tool to Evaluate Universal Web Accessibility 2000

[17] Carlos Benavídez, José L. Fuertes, Emmanuelle Gutiérrez, and Loic Martínez Semiautomatic Evaluation of Web Accessibility with HERA 2.0 CHI '05

[18] Nadia Fernandes and Rui Lopes and Luis Carrico On Web Accessibility Evaluation Environments W4A '11

[19] Nadia Fernandes and Daniel Costa and Sergio Neves and Carlos Duarte and Luis Carrico Evaluating the accessibility of Rich Internet Applications. W4A '12

[20] W3 Clean up your Web pages with HTML TIDY http://www.w3.org/People/Raggett/tidy

[21] Hironobu Takagi and Takashi Itoh and Shinya Kawanaka and Masatomo Kobayashi and Chieko Asakawa Social Accessibility: Achieving Accessibility through Collaborative Metadata Authoring ASSETS '08

[22] Yeliz Yesilada and Giorgio Brajnik and Markel Vigo and Simon Harper Understanding web accessibility and its drivers. W4A2012 - Technical, April 16-17, 2011

[23] Lisa Seeman The Semantic Web, Web Accessibility, and Device Independence W4A '04

[24] Arnaud Jasselette, Marc Keita, Monique Noirhomme-Fraiture, Fré dé ric Ran-dolet, Jean Vanderdonckt, Christian Van Brussel and Donatien Grolaux Automated Repair Tool For Usability And Accessibility Of Web Sites INTERACT '05

[25] Yuquan Shi The accessibility of Chinese local government Web sites: An exploratory study Government Information Quarterly Volume 24, Issue 2

[26] Laura O'Grady and Laurie Harrison Web accessibility validation and repair: which tool and why 2003

[27] Julio Abascal, Myriam Arrue, Inmaculada Fajardo, Nestor Garay and Jorge Tomas The use of guidelines to automatically verify Web accessibility Universal Access in the Information Society Volume 3, Issue 1

[28] Silvia Mirri, Ludovico A. Muratori and Paola Salomoni Monitoring Accessibility: Large Scale Evaluations at a Geo-Political Level ACM 2011

[29] Said Talhi, Fairouz Khadraoui and Mahieddine Djoudi Implementing WAI Authoring Tool Accessibility Guidelines in Developing Adaptive Elearning IJMECS Vol.4, No.9

[30] Grace Mbipom and Simon Harper The transition from web content accessibility guidelines 1.0 to 2.0: what this means for evaluation and repair SIGDOC '09

[31] David Sloan, Andy Heath, Fraser Hamilton, Brian Kelly, Helen Petrie and Lawrie Phipps Contextual web accessibility - maximizing the benefit of accessibility guidelines ACM '06

[32] Shadi Abou-Zahra WAI-ACT: web accessibility now WWW '12