UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

EWA - EVALUATING WEB ACCESSIBILITY

Nádia Raquel Palma Fernandes

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Sistemas de Informação

2011

UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

EWA - EVALUATING WEB ACCESSIBILITY

Nádia Raquel Palma Fernandes

DISSERTAÇÃO

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

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Sistemas de Informação

2011

Acknowledgements

I would lite to thank who made this project possible, my adviser Prof. Doutor Luis Carriço and my informal advisor Rui Lopes. I am grateful to both for guidance and for encouraging me to do better when necessary.

I thank those who always supported me unconditionally, without wanting anything in return, and taught me the values and all you can ask of parents. Thanks.

I thank those who had been by my side in these last few years with whom I have great adventures in college, the eternal group, João and Tiago. We will forever be a group.

To the best friends that I could have, João, Tiago, Tânia, Miguel, Diogo, thank you for all the support and understanding and for all the good times we had together.

Finally, dear friend thank you and see you soon.

To my family and friends.

Resumo

A Web, como uma plataforma aberta para a produção e consumo de informação, é usada por vários tipos de pessoas, algumas com determinadas incapacidades. Os sítios Web devem ser desenvolvidos tendo em conta que a informação deve ser compreendida por todos, isto é, deve ser acessível. Para analisar se uma determinada página Web é acessível, é necessário inspeccionar as suas tecnologias de front-end (por exemplo: HTML, CSS, Javascript) esta inspecção pode ser feita de acordo com regras específicas. Um processo de avaliação interessante diz respeito à utilização de ferramentas de acessibilidade que automaticamente inspeccionam uma página Web.

A avaliação automática de acessibilidade pode ocorrer em vários ambientes de execução e pode ser realizada em HTML original ou transformado. O HTML original é o documento HTML inicial derivado do pedido HTTP. O HTML transformado resulta da aplicação das tecnologias de front-end no HTML original, como realizado pelo CSS e pelo Javascript/Ajax. Isto pode alterar substancialmente a estrutura do conteúdo, apresentação e capacidade de interacção propiciada por uma determinada página Web. Esta distinção entre as versões do HTML original e transformado de uma página Web é fundamental, porque é o HTML transformado que é apresentado e com que os utilizadores interagem no Web browser.

Os processos existentes de avaliação automática, como os apresentados em [35, 34, 37], normalmente ocorrem no HTML original. Desta forma, as conclusões sobre a qualidade da acessibilidade de uma página Web podem estar erradas ou incompletas. Neste trabalho realizou-se uma framework de avaliação de acessibilidade Web em diferentes ambientes, com o objectivo de compreender as suas semelhanças e diferenças a nível de acessibilidade.

A arquitectura da framework de avaliação consiste em quatro principais componentes: *Execution Environments*, *QualWeb evaluator*, *Techniques* e *Formatters*.

O *QualWeb evaluator* é responsável por realizar a avaliação da acessibilidade na página Web usando os recursos fornecidos pelo componente das *Techniques*, que usa o componente *Formatters* para adequar os resultados em formatos de serialização específicos, tais como relatórios de erros. O *QualWeb evaluator* pode também ser usado independentemente dos vários em diferentes ambientes de execução (*Execution Environments*).

Os *Execution Environments* são responsáveis pela transformação do documento HTML de uma página Web na sua representação equivalente numa árvore HTML DOM.

O componente *Techniques* contém as técnicas de avaliação do front-end, optando-se por usar W3C WCAG 2.0 [17], porque é um dos mais importantes padrões de acessibilidade.

A arquitectura foi pensada de forma a permitir a serialização dos resultados da avaliação em qualquer formato. Assim, as bibliotecas de formatação estão contidas dentro do componente *Formatters*. Foi utilizada a serialização EARL [9], porque é um formato padrão para relatórios de acessibilidade. Os resultados obtidos podem ser interpretados por qualquer ferramenta que use este formato, permitindo comparar os resultados desta ferramenta com os de outras. A qualquer altura pode ser adicionado outro tipo de formatação nos Formatters (por exemplo, relatórios em PDF).

O componente *Execution Environments* representa os vários ambientes de execução e foram usados dois tipos: o *Command Line* e o *Browser*. O *Command Line* é o equivalente ao ambiente de execução normalmente utilizado para realização de testes automáticos, ou seja, o ambiente que fornece o HTML original. O *Browser* é o ambiente de exevuçao onde o HTML usado é o transformado.

A arquitectura foi desenvolvida de forma a ser flexível e modular, sendo possível a qualquer momento a adição um novo módulo dentro dos componentes principais. Por exemplo: adição de um novo ambiente de execução, ou outro tipo de técnicas.

Para se conseguir avaliar da mesma forma os ambientes de execução, a implementação foi realizada na linguagem de programação Javascript, porque é facilmente suportada nos dois ambientes. Esta implementação permite o estudo comparativo das diferenças da avaliação da acessibilidade Web em ambos.

Foi também desenvolvida uma bateria de testes para se validar de forma sistemática as técnicas implementadas nos dois ambientes. Desta forma, os resultados obtidos para cada técnica foram validados, antes de o avaliador ser utilizado para testes mais complexos. Garantindo que os resultados obtidos posteriormente estariam correctos.

Finalmente, foi realizado um estudo para se perceber se era realmente mais vantajosa a realização de avaliações de acessibilidade sobre o documento HTML transformado, em vez de no original. Foi avaliado um conjunto de páginas Web nos dos ambientes implementados. Com a comparação dos resultados obtidos nos dois ambientes conclui-se: que são detectados muito mais elementos no Browser e com isso conseguem-se obter mais resultados de acessibilidade neste ambiente; e que há uma diferença muito significativa na estrutura do HTML transformado e original. Pode assim afirmar-se, que há uma mais-valia significativa na realização deste tipo de avaliação de acessibilidade no Browser.

No entanto, é importante considerar que as páginas Web são frequentemente compostas por templates. Os templates são adoptados para manter a uniformidade de distribuição, para tentar melhorar a navegação dos sítios Web e para manter objectivos das marcas.

Hoje em dia, o desenvolvimento da Web é muito centrado na utilização de templates para facilitar a coerência, a implementação e a manutenção de recursos de um sítio Web. Foi determinado que 40-50% do conteúdo da Web são templates [23]. Apesar desta ampla utilização de templates, as avaliações de acessibilidade avaliam as páginas como um todo, não procurando similaridades que se verificam devido à utilização dos templates. Esta forma de avaliação das páginas com um todo, faz com que os verdadeiros resultados de acessibilidade fiquem diluídos no meio de um grande número de resultados repetidos.

Contudo, os templates podem ser uma mais-valia para que faz um sítio Web, não sendo necessário corrigir o mesmo erro várias vezes, basta corrigi-lo uma vez que o próprio template propaga essa correcção por todo o sítio Web.

Realizou-se por isso um algoritmo de detecção de templates, utilizando como base um algoritmo de detecção de matching já existente [14]. Este algoritmo detecta similaridades entre duas árvores HTML DOM.

Para se perceber concretamente as semelhanças nos elementos HTML entre as páginas Web, efectuou-se um estudo para detecção dos templates em vários sítios Web. O processo utilizado consistiu nos seguintes passos: 1) detectar os templates entre várias páginas do mesmo sítio Web; 2) proceder à avaliação das páginas usando o nosso avaliador definido no inicio do trabalho; e finalmente, 3) separar os ficheiros EARL obtidos em dois ficheiros, um que continha a parte comum entre duas páginas e outro que continha a parte especifica, *template set* e *specific set*, respectivamente. Desta forma, determinou-se que aproximadamente 39% dos resultados de acessibilidade foram verificados nos templates. É uma percentagem bastante elevada de erros que pode ser corrigida de uma só vez.

Com este trabalho foi então realizado: uma análise comparativa dos dois ambientes de execução; um algoritmo de detecção de templates que permitiu a criação de uma nova métrica de acessibilidade, que quantifica o trabalho necessário para reparar problemas de acessibilidade e que pode até ser utilizada como auxiliar de outras métricas; a arquitectura de um sistema de avaliação que pode ser executado em vários ambientes; um avaliador de acessibilidade Web baseado em WCAG 2.0, genérico o suficiente para permitir a utilização de quaisquer técnicas, formatadores ou ambientes de execução que se pretenda; e uma bateria de testes que permite a verificação dos resultados de acessibilidade da avaliação, de acordo com as técnicas escolhidas.

Palavras-chave: Acessibilidade Web, Avaliação Automática, Templates de páginas Web

Abstract

The purpose of this work was to improve the automated Web accessibility evaluation, considering that: evaluation should target what the end users perceive and interact with; evaluation results should address accessibility problems in a focused, uncluttered, way; and results should reflect the quality adequately to the stakeholders.

These considerations had the following goals: analyse the limitations of accessibility evaluation in two different execution environments; provide additional guidance to the developer in order to correct accessibility errors, that considers the use of templates in page development and avoid cluttering the relevant evaluation results; and define evaluation metrics that reflect more adequately the difficulty to repair Web sites' problems.

An accessibility evaluator, *QualWeb*, was implemented and it performs W3C WCAG 2.0 evaluations. Unlike most existing automatic evaluators, this approach performs evaluations on the HTML documents already processed, accessing content as presented to the user. The evaluator also allows the evaluation on unprocessed HTML documents, as traditionally done. The framework was designed to be flexible and modular, allowing easy addition of new components. The serialization chosen was EARL that can be interpreted by any tool understanding this standard format.

To verify the correctness of the WCAG techniques implementation, a control test-bed of HTML documents was implemented, representing the most significant problems that should be detected. Results of the first experimental study confirmed that there are deep differences between the HTML DOM trees in the two types of evaluation. This shows that traditional evaluations do not present results coherent with what is presented to the users.

It was also implemented a template detection algorithm allowing the adequate detailed and metric-based reporting of an accessibility evaluation. This form of reporting can be used by existing tools, which can become more helpful in producing accessible Web sites. Results from the second experimental study show that template-awareness may simplify assessment reporting, and approximately 39% of the results are reported at least twice, of which approximately 38% are errors that can be corrected once.

Keywords: Web Accessibility, Automatic Evaluation, Web page templates

Contents

List of Figures **xviii**

List of Tables **xxi**

1 Introduction **1**
- 1.1 Work Context . . . 1
- 1.2 Objectives . . . 2
- 1.3 Work Plan . . . 3
  - 1.3.1 Description of the Tasks . . . 4
- 1.4 Contributions and Results . . . 5
- 1.5 Publications . . . 6
- 1.6 Institutional Context . . . 6
- 1.7 Document Structure . . . 7

2 Requirements and Related Work **9**
- 2.1 Web and Browsing . . . 9
  - 2.1.1 Web Browser Process . . . 9
- 2.2 Web Accessibility Evaluation . . . 11
  - 2.2.1 Accessibility Standards . . . 12
  - 2.2.2 Validation Corpus . . . 14
  - 2.2.3 The Evaluated Material . . . 14
- 2.3 Using, Ensuring and Developing the Accessible Web . . . 14
  - 2.3.1 Reporting Standards . . . 15
  - 2.3.2 The Impact of Templates . . . 15
  - 2.3.3 Metrics . . . 18
- 2.4 Existing tools . . . 20
- 2.5 Summary and Requirements . . . 21

3 Evaluation Framework **23**
- 3.1 Architecture . . . 23
- 3.2 Execution Environments . . . 25
  - 3.2.1 Command Line Environment . . . 28

3.2.2 Browser Environment 28
3.3 QualWeb Evaluator 29
3.3.1 QualWeb Evaluator Client 30
3.3.2 QualWeb Evaluator Server 30
3.4 Techniques 31
3.4.1 WCAG 2.0 31
3.5 Formatters 35
3.5.1 EARL 35
3.6 Template-based Evaluation 37
3.6.1 Fast Match algorithm 38
3.6.2 A Template-Aware Web Accessibility Metric 39
3.7 Implementation details 40
3.8 Summary 41

4 Evaluation **43**
4.1 Validation of WCAG 2.0 Techniques Implementation 43
4.2 Experimental Study 1 - Web Accessibility Evaluation 45
4.2.1 Setup 45
4.2.2 Data Acquisition and Processing 46
4.2.3 Results 47
4.2.4 Discussion 52
4.2.5 Limitations 53
4.3 Experimental Study 2 - Templates on Web Accessibility Evaluation . . . 54
4.3.1 Setup 54
4.3.2 Data Acquisition and Processing 55
4.3.3 Results 55
4.3.4 A Template-Aware Web Accessibility Metric 55
4.3.5 Discussion 57
4.3.6 Limitations 58
4.4 Summary 58

5 Conclusion **61**
5.1 Future Work 62

A Papers Written **63**
A.1 Avaliação Pericial de Barreiras ao Acesso sobre Sítios Web de Entidades Públicas - Interacção 2010 63
A.2 On Web Accessibility Evaluation Environments - W4A 2011 66
A.3 An Architecture for Multiple Web accessibility Evaluation Environments - HCII 2011 77

A.4 The Role of Templates on Web Accessibility Evaluation - Assets 2011 . . 88

Abbreviations **91**

Bibliography **96**

List of Figures

2.1 Web Browsing Resource Interaction 10
2.2 Web Page Loading Process 11
2.3 EARL example 16
2.4 Template example 17
2.5 Typical Web page template structure 17

3.1 Architecture of the Evaluation Framework 24
3.2 QualWeb evaluator sub-modules. 25
3.3 Flowchart of assessment in the Command Line execution environment. . . 26
3.4 Flowchart of the sequence of assessment in the Browser execution environment 27
3.5 Evaluation execution example on Browser 28
3.6 Function to obtain the HTML document of the presented Web page. . . . 29
3.7 Scheme of the array of results. 30
3.8 Scheme of the new representation of the results. 30
3.9 Excerpt from WCAG 2.0 H64 technique 34
3.10 Example of Node-Template module application. 35
3.11 EARL document 36

4.1 Number of Test Documents per Technique 44
4.2 A HTML test document with an example of the right application of technique H25. 45
4.3 A HTML test document with an example of the wrong application of technique H25. 45
4.4 Comparing size in bytes in both execution environments 47
4.5 Comparing size in HTML Elements count in both execution environments 47
4.6 Number of HTML Elements that Passed 48
4.7 Number of HTML Elements that Failed 48
4.8 Number of HTML Elements that had Warnings 48
4.9 Browser vs Command Line per criterion (log-scale on HTML Elements count) 49
4.10 Browser vs Command Line for criterion 1.1.1 51

4.11 Browser vs Command Line for criterion 1.2.3 51
4.12 Browser vs Command Line for criterion 2.4.4 51
4.13 Browser vs Command Line for criterion 3.2.2 51
4.14 Applicability of WCAG 2.0 techniques on one of the evaluated Web pages. 55
4.15 Graphs represent the elements per Web page, on top row left to right the Web sites are: *Google*, *Público*, *DN* and *Wikipedia*. In bottom row the Web site are: *Facebook*, *Amazon* and *Wordtaps* 56

List of Tables

1.1 Initial Schedule 4
1.2 Revised Schedule 4

3.1 Techniques Implemented 32
3.2 Criteria Considered 33

4.1 False positives and false negatives in criteria applicability on *Command Line* execution environment 50
4.2 Analysed Web sites 54

Chapter 1

Introduction

1.1 Work Context

The Web, as an open platform for information production and consumption, is being used by all types of people, with miscellaneous capabilities, including those with special needs. Consequently, Web sites should be designed so that information can be perceived by everyone, i.e., should be accessible.

The importance of Web accessibility is increasing in the international context, and especially in the European Union. In Europe, more and more countries have legislation requiring that public Web sites have to be accessible [21]. In Portugal, the directive that requires the accessibility of institutional Web sites is the Council of Ministers Resolution number 155/2007 [31], which uses the technicalities specified in W3C WCAG 1.0 [15]. According to this directive, "the organisation and presentation of information provided by the Internet sites of public sector should be chosen to allow or facilitate access for citizens with special needs" [31]. Web accessibility should cover at least the information relevant to understand the content. WCAG [8, 17] is one of the most used technical standards to accessibility evaluations.

From the different ways that Web page inspection can be done, an interesting evaluation procedure concerns the usage of accessibility assessment software tools that algorithmically inspect a Web page's structure and content in an automated way. Considering the amount of information on the Web, this approach surely has properties not shared by manual evaluations.

Traditionally, accessibility evaluation software assesses source-documents, i.e., HTML as produced by IDE or developers. However, often the Web browser transforms those documents before present them to the user. The result, after the application of CSSs and JavaScript, for example, can be substantially different from the original. Ultimately, it is transformed HTML that is presented and interacted by all users within a Web browser. Consequently, conclusions over the accessibility quality of a Web page based on source analysis can be incomplete, or, in extreme erroneous. It is therefore important to access

the transformed HTML documents and understands how deep the differences towards the original documents are.

Front-end Web development is highly centred on the use of templates to ease implementing and maintaining coherence of Web site structural features. Because of that, evaluating Web sites could lead to misleading accessibility evaluation results, i.e., the same errors are repeated over and over obfuscating the final reports. This exacerbates the repairing problems, when they occur in a template, and dilute the remanding ones within the numerous reported errors. While managing repairing processes, this may simply kill the corrective project (too demanding) or difficult the distribution of correction tasks (several developers correcting the same problem).

With template-aware accessibility evaluation tools, developer teams can better manage the accessibility repair process and have a more realistic perspective of the actual effort necessary to do it.

In order to effectively evaluate accessibility considering templates, one should first assess if the amount of errors found in common elements amongst Web pages is relevant in respect to the total amount.

The existent Web accessibility metrics that use these reports consider a great amount of repeated accessibility evaluation results spread across evaluated Web pages, caused by templates. This way, new metrics have to be created that reveal the real accessibility quality of the Web pages/sites.

1.2 Objectives

The purpose of this work is to improve the automated Web accessibility evaluation. This purpose, wide as it can be, is focussed in this thesis in finding the adequate evaluation target and results to the right stakeholders. Three basic considerations are taken into account: the evaluation should target what the end users perceive and interact with; the evaluation results should address accessibility problems in a focused, uncluttered, way; global results of evaluation should reflect the quality in terms adequate to the stakeholders.

Given this summary, the main goals of this work are:

- discover the limitations of accessibility evaluation in the two different execution environments, before and after browse processing, by achieving a set of concrete results and well-grounded assessment of the differences between the two and their advantages and disadvantages in the accessibility evaluation of Web pages;

- provide additional guidance to the developer and the developers' team management in order to correct accessibility errors, that considers the use of templates in page development and avoid cluttering the relevant evaluation results;

- define evaluation metrics that reflect more adequately the difficulty to repair Web sites' problems.

For that the following technical objectives are aimed:

- design a flexible and modular architecture for a framework of accessibility evaluation, allowing the addition of new evaluation techniques and different formatters;

- implement a set of WCAG 2.0 techniques [12] that have automatable properties and which will allow to make the comparative analyses between execution environment;

- implement a serialisation of EARL. This way, it can be interpreted by any tool understanding this standard format, and it allows the comparison between these results and others obtained with other evaluation tools;

- adapt and implement a template detection algorithm that allows the adequate detailed and metric-based reporting of a Web site accessibility evaluation;

- implement a framework for Web accessibility evaluation that incorporates the above technical objectives.

To verify the correctness of the WCAG implementation, a control test-bed of HTML documents was implemented, that represents some of the most significant problems that should be detected by WCAG techniques. That test-bed is being developed as a part of a large reference corpus that will constitute the W3C standard for WCAG implementations assessment.

1.3 Work Plan

In the beginning of this work, it was defined a set of the major tasks and it was defined its schedule. Table 1.1 contains the expected duration (in months) of the major tasks, with starting date in September and the supposed end date in June. The expected durations were estimated considering a little more time for each task. This way, a delay in a task completion does not prejudice other tasks which depend on this one. Besides, some of these tasks needed to be done sequentially and another task could be done in parallel.

The first six tasks were performed faster than expected, because of that it was decided to add a few more tasks. In this new schedule (Table 1.2) *tasks 2-6* have the real duration.

The revised plan was almost completely accomplish in time, only the last task (*task 9*) took a little longer than expected (1 month). The reason was the writing of a paper to ASSETS 2011 and of the proposal for a doctoral scholarship application to FCT.

Table 1.1: Initial Schedule

Tasks	Expected duration (months)
1. Related work study	1
2. Design a battery of test cases	1.5
3. Preliminary Report	0.75
4. Design and perform the architecture of evaluation system	1.5
5. Implementation of prototypes	1.25
6. Comparative analysis of test cases	1
7. Writing of the thesis	2

Table 1.2: Revised Schedule

Tasks	Revised duration
1. Related work study	1.5
2. Design a battery of test cases	0.75
3. Preliminary Report	0.75
4. Design and perform the architecture of evaluation system	1
5. Implementation of prototypes	0.5
6. Comparative analysis of the test cases	0.5
7. Implementation of a template detection algorithm	1
8. Experimental study	1
9. Writing of the thesis	2

1.3.1 Description of the Tasks

Task 1 - Related Work Study

Study and analysis of the existing work/state of Web accessibility evaluation. It was performed in the beginning of the work and it was revised in the end to verify if there have been changes in the state of the art. Besides of the study of Web accessibility evaluation state of the art (which lasted 0.5 months), it was accumulated the study and analysis of the Web page template detection.

Task 2 - Design a battery of test cases

Design and perform a battery of test cases for WCAG 2.0 accessibility to different Web technologies (HTML, CSS, Javascript) to verify the accuracy of the techniques implemented.

Task 3 - Preliminary Report

This task happened after all the related work was performed and it summarizes the state of the work, the goals and the work that still needed to be done.

Task 4 - Design and perform the architecture of evaluation system

It is a very important part of this work and it consisted in the design and development of the Web accessibility evaluator. This task was done simultaneously with *task 2*.

Task 5 - Implementation of prototypes

Implementation of two prototypes for WCAG 2.0 evaluation in the following runtime execution environments: Command Line and Browser.

Task 6 - Comparative analysis of test cases

This task consisted in a comparative analysis of the test cases in different execution environments.

Task 7 - Implementation of a template detection algorithm

This algorithm was used to detect Web page templates in the Web pages. This task was the previous *task 7* that skipped to *task 9*.

Task 8 - Experimental study

This task consisted in test the template detection algorithm in some Web sites and a new Web accessibility metric was created.

Task 9 - Writing of thesis

The final task which occurred mostly after the experiments and the results are analysed. However, some chapters were written during the work.

1.4 Contributions and Results

The work performed in the scope of the PEI was:

- a comparative analysis of both execution environments, to conclude the differences between them and the advantages of perform Web accessibility evaluation in Web browser context;
- a template detection algorithm which allowed the creation of accessibility metric that allows the quantification of the necessary work to repair the accessibility problems;
- an architecture of an evaluation system that allows evaluations in several execution environments;

- a Web accessibility evaluator based on WCAG 2.0 and generic enough to allow the use of other techniques, formatters or execution environments;
- a test-bed that allows to verify the accessibility results of an evaluation, according with the WCAG 2.0 techniques.

1.5 Publications

During this work, the following papers were produced:

- a poster was accepted in the conference Interacção 2010, with the title "Avaliação Pericial de Barreiras ao Acesso sobre Sítios Web de Entidades Públicas". This paper details a study of the accessibility barriers found in a public Web site (Appendix A.1);
- a full paper accepted in the conference W4A 2011 (8th International Cross-Disciplinary Conference on Web Accessibility). The percentage of acceptance rate was 33%. The title is "On Web Accessibility Evaluation Environments" and the paper details an experimental study designed to understand the differences posed by accessibility evaluation in the Web browser (Appendix A.2). This paper resulted in an invitation to submit the paper in a Special Edition of the New Review of Hypermedia and Multimedia (NRHM) Journal, published by Taylor & Francis. That will bring together some of the best work presented at this year's W4A conference;
- a full paper accepted in the conference Human Computer Interaction International 2011 (the 6th International Conference on Universal Access in Human-Computer Interaction), with the title "An Architecture for Multiple Web Accessibility Evaluation Environment". This paper describes the architecture that was used in this work (Appendix A.3); and
- a short paper, under evaluation, submitted for poster presentation to the conference ASSETS 2011 (the 13th International ACM SIGACCESS Conference on Computers and Accessibility), with the title "The Role of Templates on Web Accessibility Evaluation". The paper contains a preliminary study that demonstrates that a significant part the accessibility errors found in relevant Web pages occur in templates (Appendix A.4).

All the papers can be consulted in the appendix section.

1.6 Institutional Context

The master degree project in computer engineering — PEI -– was conducted in the Large-Scale Informatics Systems Laboratory – LASIGE – of Informatics Department of Faculty

of Sciences of the University of Lisbon — FCUL.

FCUL is one of the units that comprise the University of Lisbon. It has available to the student a set of modern infrastructure in order to give the students all the conditions for teaching excellence. It has more than 24 units of R&D funded and evaluated by the Foundation for Science and Technology, and engaged in multiple areas.

The R&D units at FCUL have obtained high rankings in panels of international assessment, continuing the tradition of scientific quality of the school and the increasing affirmation at the international level. Its funding is provided by FCT, the European Union or by Research for companies and government agencies.

FCUL is leading national participants in the European Programmes for Research and Technological Development and it is a partner of international cooperation agreements established by Portugal with U.S. universities. Besides, it promotes bilateral and multilaterals scientific collaboration and the link between research and industry and entrepreneurship.

1.7 Document Structure

This document is organized with the following structure:

- Chapter 2 - *Related Work*: this chapter details an assessment of the state of the art in Web accessibility evaluation, presenting what still need to be done in the area;
- Chapter 3 – *Evaluation Framework*: this chapter explains the approach chosen to each component of the framework; and details its design and implementation;
- Chapter 4 – *Evaluation*: this chapter describes how the framework is validated, the two studies performed, and all the results are presented and discussed;
- Chapter 5 – *Conclusion*: chapter that closes this report, it contains the conclusions of the work and the future work that can be done in this area, using what has been done.

Chapter 2

Requirements and Related Work

This chapter covers the main topics necessary to understand this work, being Web Accessibility Evaluation (WAE) its main theme. The text begins by describing the basic concepts of the Web and Web browsing that sustains the arguments of post processing evaluation. Then, the fundamental concepts of WAE are presented, including an explanation of the most relevant standards. Afterwards, the different usage perspectives were studied and the roles of templates and metrics were considered on the building of those perspectives.

In view of the exposed concepts and requirements, the following section analyse the most relevant existing tools. The chapter ends with a synopsis, pointing the fundamental technical components that emerge from the requirements.

2.1 Web and Browsing

In the past, the predominant Web technologies were HTML and CSS, which resulted in "static" Web pages. Today, Web is growing and on top of these technologies, newer technologies emerge. Javascript is a relevant one, if not the most.

Consequently, the Web is becoming more and more dynamic. User actions and/or automatically triggered events can alter a Web page's content. The presented content can be substantially different from the initially received by the Web browser. Because of that, it is important to understand how the Web browser works, with all the technologies, which will be explained in the next section.

The following sections describe the Web browser process and WAE.

2.1.1 Web Browser Process

The dynamics of Web pages centres around a sequence of communication steps between the Web browser and the Web servers, as depicted in Figure 2.1.

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

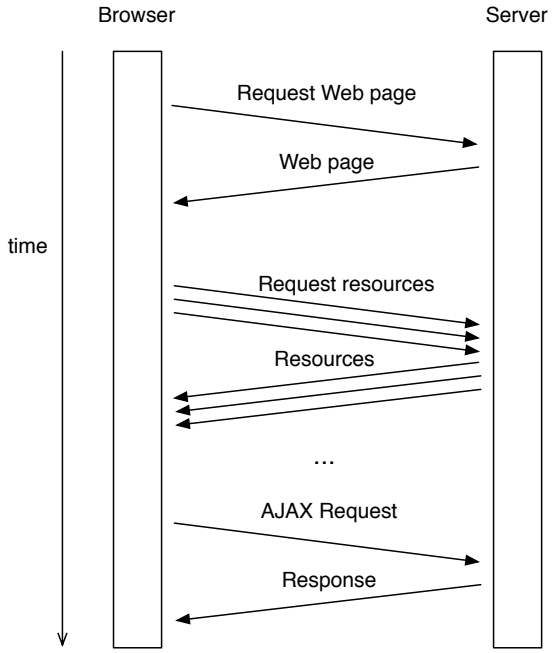

Figure 2.1: Web Browsing Resource Interaction

- *Web page*: this is the main resource that defines the skeleton of the content that will be presented in the Web browser;
- *Resources*: these complementary assets include images and other media, style sheets, and scripts that are explicitly specified in the Web page's structure (with proper HTML elements);
- *AJAX*: these resources are transmitted during or after the browser triggers the loading events for a Web page.

As a consequence, the final outcome presented in the Web browser is a mixture, supported by the architecture of the Web (*request-response* nature, of the original *Web pages* and *Resources*) and the Web page loading process within a Web browser (e.g., *AJAX*). In the next section these aspects will be detailed.

Architecture of the Web

The architecture of the Web [27] is composed by servers, URIs, and user agents. User agents (such as Web browsers) communicate with servers to perform a retrieval action for the resource identified by the URI. A server responds with a message containing a resource representation. As depicted in Figure 2.1, in the case of Web browsers, a Web page is represented not just by its HTML content, but also by a set of ancillary resources. Due to this increased complexity on handling resources and their representation for users, Web browsers process all the resources through adequate technologies (e.g., executing script), which results in the transformed HTML document that is presented to users.

Web Page Loading Process

After all resources are successfully delivered to the Web browser, four steps are sequentially executed before users are able to interact with the Web page, as depicted in Figure 2.2:

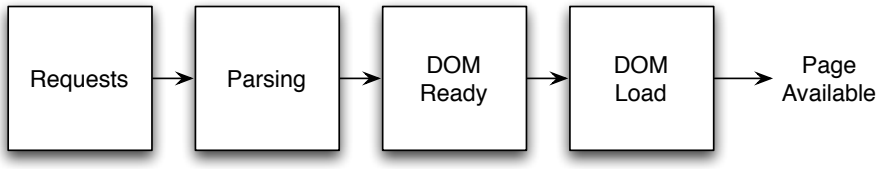

Figure 2.2: Web Page Loading Process

The first step in the Web page loading process, *Requests*, concerns on getting all resources that compose the Web page. After that, the Web browser *parses* these resources, building the HTML DOM tree (i.e., a Document Object Model is a model of how the various HTML elements in a Web page are related to each other. So the HTML document is represented as tree, in which each HTML element is a branch or leaf, and has a name [25, 26].), the "visual layout" using CSS, and constructing the execution plan based on the existing scripted behaviours. Afterwards, the Web browser triggers two events in sequence: *DOM Ready* and *DOM Load*. The former is triggered when the HTML DOM tree is ready, whereas the second is triggered after all resources are ready (e.g., CSS, images, etc).

Web pages typically 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 potentiate the addition, removal or transformation of this tree. Consequently, the Web page presented to the user might be heavily different from the URI's resource representation, which is initially transmitted to the Web browser from the Web server.

2.2 Web Accessibility Evaluation

Web access is nowadays such an important asset that is considered a fundamental right for everyone. Among the possible users, an extensive group of people, access the Web through assistive technology (AT), because of their disabilities. These technologies articulate with Web browsers (or user agents) to convey the Web page content in an adequate way to each individual. Therefore, it is paramount that Web content is produced in a way that is compatible with those technologies and adequate to the different disabilities.

To assess and improve that compatibility and adequacy, the first step should be to evaluate. WAE is, thus, an assessment procedure to analyse how well the Web can be used by people with different levels of disabilities [24]. Unfortunately, current studies

show that many Web sites still cannot be accessed in the same conditions, by a large number of people [24, 31]. This is an important issue, which motivates further work to be done, in this area. Besides, additional dissemination and adequacy improvement of the tools and means available to evaluate and report, and later correct, must be done to improve accessibility quality.

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. They provide an important opinion to discover the accessibility problems and to find their solutions. However, assessment by users is often subjective. Frequently, when they cannot perform a task that does not mean they found an accessibility problem, it can be a problem in the AT, Web browser or other. Therefore, these problems are very difficult to filter, so the majority of them cannot be generalized. 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. Not being a substitute to the users' based evaluation is a very important complement. However, it is a time-consuming process too, and it can bring potential bias in the comparison of the different Web pages' accessibility quality [24, 31].

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 big benefits are scalability and objectivity [31]. However, it has limitations that 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. Besides, these guidelines should be applied in the rendered or transformed state of a Web page/site. In the next section, it will be presented the most relevant accessibility guidelines standards.

2.2.1 Accessibility Standards

WCAG is one of the most used technical standards to accessibility evaluations and to encourage creators (e.g., designers, developers) in constructing Web pages according to a set of best practices. This standard covers a wide range of recommendations for making Web content more accessible. WCAG 1.0 was published in 1999, and tackled the technical constraints of the Web as it was. With the evolution of Web standards, such as HTML, and how developers and designers explore Web technologies as of today, WCAG 1.0 is

often seen as outdated. Therefore, WCAG 2.0 was created as response to this evolution, thus, allowing developers and designers to evaluate accessibility concerns in a more realistic setting. If this standard is used, a good level of accessibility can be guaranteed [24, 31].

WCAG 2.0 contains 12 guidelines chosen to address specific problems of people with disabilities. These guidelines provide the goals that should be used to make Web content more accessible. Each guideline has testable success criteria [18]. Some examples of WCAG 2.0 are [17]: provide text alternatives for any non-text content so that it can be changed into other forms people need, such as large print, braille, speech, symbols or simpler language; make all functionality available from a keyboard; provide ways to help users navigate, find content, and determine where they are; help users avoid and correct mistakes of input, etc.

The guidelines and success criteria are grouped into four principles, which promote the foundation for Web accessibility:

1. *perceivable* - information and user interface components must be apprehended by users;

2. *operable* - user interface components and navigation must have easy interaction for users;

3. *understandable* - information and the operation interface must be easy to comprehended by the users;

4. *robust* - users must be able to access content using a wide variety of user agents.

These principles have to be accomplished in order that a user with disabilities is able to use the Web.

To help developers effectively implement the success criteria, WCAG 2.0 techniques [16] were created. These techniques support the success criteria and describe the basic practices applicable to any technology or to a specific technology. Consequently, the Web page accessibility evaluation is ultimately held by technique. The evaluation outcomes by technique – *applicability* of the techniques – can be:

- *Fail* or *pass* – 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 e W3C recommendations for the techniques.

Some examples of the techniques for HTML are: providing text and non-text alternatives for object; Combining adjacent image and text links for the same resource; etc.

2.2.2 Validation Corpus

The accessibility standards can be performed in many different ways and in many programming languages. Several implementations of WCAG 1.0 and not so many of 2.0 exist. As in other areas, though, the correctness of the implementation should be assessed in regards to the interpretation of the guidelines. For that, usually, a corpus of validation, or a test-bed, is produced and initially humanly validated. That is frequently an enormous task. Then each implementation of the guidelines is applied on that test-bed to assess its validity. W3C started the production of that test-bed for WCAG 2.0 (WCAG 2.0 Test Samples Development Task Force (TSD TF) Work Statement) [7], but it is still an on-going work.

2.2.3 The Evaluated Material

Automatic WAE relies on the application of WCAG 2.0 techniques on the elements of a Web page in order to assess the quality that it is presented to the end-users. Thus, WCAG has evolved considering new aspects of technology. It is widely accepted that automatic WAE tools should adopt the latest WCAG recommendations.

However, one should also question the evaluation target. Traditionally, it has been the source documents that are returned on the first HTTP request. Yet, nowadays, Web pages are mostly dynamic. As described above, what is presented to the user is often very different from what is obtained in that request. Thus, it is paramount that the WAE tools also evolve on the material that is assessed, targeting the rendered or transformed HTML.

2.3 Using, Ensuring and Developing the Accessible Web

It is important to understand who are the stakeholders to use automatic WAE and for what purpose. Three types of stakeholders can be identified [30]: final users (i.e., those not experts exploring the Web), public bodies (i.e., those who oversee the enforcement of laws concerning the accessibility of Web sites) or developers.

Final users may want to perform an accessibility evaluation before entering a Web site/page. This way, they assess if it is worth to explore, return later or if they should look for alternatives. The time spent looking for information on a Web site that does not provide it adequately can be large and frustrating. Final users can also use WAE tools to alert the Web site producers to the lack of accessibility or to disseminate to communities its quality or lack of it.

Public bodies may want to perform an accessibility evaluation to verify if the legislation of Web accessible is being respected. In Portugal, the directive that requires the accessibility of institutional Web sites is the Council of Ministers Resolution number 155/2007 [31]. European Commission has a new initiative, *European i2010 initiative on*

e-Inclusion, which includes a "strategy to improve accessibility to the Information Society for all potentially disadvantaged groups" [4].

These two types of stakeholders need metrics to understand the level of accessibility. W3C metrics could be used and sometimes are enforced. Selective metrics, e.g. specific disability oriented, can also be important for different communities of end users. Public bodies may also need simple error reporting, to understand the gravity of errors and if possible the correction effort required.

Developers and developer teams need the results of accessibility evaluations to correct the Web pages that they developed. Their problem is not as much to understand how difficult it is to browse the Web page or site, but to grasp the amount of effort they have to spend in doing it. For that, it would be interesting to have metrics that access the development effort. Similarly, error reporting should consider the development process, not only in the complexity of reporting but also on its standardization as a form to make designers' teams collaboration more flexible and easy, and for integration in development tools.

2.3.1 Reporting Standards

EARL [9] is a standard format for support of the evaluation of Web applications. It is a vocabulary to describe the results of test execution, to facilitate its processing and interpretation by different tools. This way, EARL must be used in accessibility evaluators, validators or other content checkers. Besides, it is expressed in RDF, which can be extended easily and be adapted to any domain, as in this case accessibility. Figure 2.3 shows an example of an EARL excerpt.

However, accessibility results can be presented in a complex way to developers since they are not accessibility experts, e.g., big reports or tools that they cannot understand. If the report is self-evident, obvious and self-explanatory to the developers, then they will understand them without problems [28].

There is a large number of automatic tools that generate a different instance for the same type of problem. These instances lead to many repetitions of the same problem in the report. Hence, a simplified list with the type of problem and one or two examples of the actual error is enough so that the designers/developer can fix it without major difficulties.

Consequently, the Web Accessibility Evaluation has to use EARL to deliver evaluation results, because it is the recommended format to present Web Accessibility Evaluation results. Besides, these reports should be simplified to facilitate developers' work.

2.3.2 The Impact of Templates

Templates are usually generated using authoring/publishing tools or by programs that build HTML pages to publish content (Figure 2.4). Some examples are: navigation side-

```
<rdf:RDF
      xmlns:earl="http://www.w3.org/ns/earl#"
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xml:base="http://www.example.org/earl/report#">

   <earl:Assertion rdf:ID="ass3">
     <earl:result rdf:resource="#error3"/>
     <earl:test rdf:resource="... /xhtml1-strict.dtd"/>
     <earl:subject rdf:resource="... /resource/index.html"/>
     <earl:assertedBy rdf:resource="#assertor01"/>
   </earl:Assertion>

   <earl:Assertion rdf:ID="ass1">
     <earl:result rdf:resource="#error1"/>
     <earl:test rdf:resource=".../xhtml1-strict.dtd"/>
     <earl:subject rdf:resource="... /resource/index.html"/>
     <earl:assertedBy rdf:resource="#assertor01"/>
   </earl:Assertion>

   <earl:Assertion rdf:ID="ass2">
     <earl:result rdf:resource="#error2"/>
     <earl:test rdf:resource="... /xhtml1-strict.dtd"/>
     <earl:subject rdf:resource="... /resource/index.html"/>
     <earl:assertedBy rdf:resource="#assertor01"/>
   </earl:Assertion>

   <-- ... -->

</rdf:RDF>
```

Figure 2.3: EARL example

bars, corporate logos in a specific location, headers or menus, locations, contact information, ads, and footers [23].

```
<?php
        include(''head.php'');
        include(''menu.php'');

        //page specific code goes here

        include(''footer.php'');
?>
```

Figure 2.4: Template example

In addition to facilitate Web page construction, templates can also increase the reuse of the Web page code and improve its structure. They maintain the uniformity of layout (Figure 2.5), try to enhance navigation of Web sites and maintain branding goals [36].

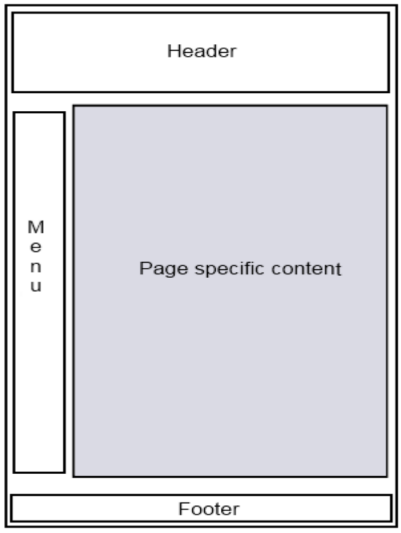

Figure 2.5: Typical Web page template structure

Templates are highly used in the modern Web development, to ease the implementing and maintaining coherence of Web site navigational and presentational features. It has been determined that 40-50% of the Web content is template content [23]. Hence, this mechanism is of high profusion throughout the Web.

Considering templates, automatic WAEs, as they are commonly done, could provide misleading results to developers, i.e., the same errors are repeated over and over obfuscating the final reports. This exacerbates the repairing problems when they occur in a template and dilute the remaining ones, within the numerous reported errors. While managing repairing processes, this may simply kill the corrective project (because it appears

as too demanding) or complicate the distribution of correction tasks (several developers fixing the same problem). With template-aware WAE tools, developer teams can better manage the accessibility repair process and have a more realistic perspective of the actual effort necessary to do it.

Solutions, like doing evaluation in the original template and sources, yields heavily distorted results and are not reasonable alternatives. First, they assess documents that are rather different from the ones presented to users. Elements are often combined in the rendered pages and errors emerge from that combination. Secondly, and reinforcing the previous, templates are frequently incomplete documents, sometimes improper for automatic evaluation.

Template detection

Template detection is often used in the fields of Information Retrieval and Web page Clustering. Towards information retrieval, template detection and removal increase precision [11] can positively impact performance and resource usage in processes of analysis of HTML pages [36]. Regarding Web page Clustering, templates could help in cluster structurally similar Web pages [13].

Search logs were used to find paths in the DOM trees that mark out important parts of pages [13]. These paths are identified by analysis of the Web site, using search data for popular pages to infer good paths.

Pagelets are also used to perform template detection. These are a self-contained logical part within a page that has a defined topic or functionality [11]. A page can contain one or more *pagelets*, depending on the number of topics and functionalities.

Although most of these works are not at the level of accessibility, it was already mentioned the importance of considering Web page templates in accessibility issues [29]. It was suggested the inclusion of accessibility on the design, using accessible content templates, to preserve accessibility [32]. Besides, template detection was used to detect some usability problems in an early stage [10]. However, this last work does not explore the specific issue of accessibility or the impact of the construction of accessible Web pages.

In conclusion, template detection can be a big advantage in simplifying accessibility reports and in the error repair. However, until now they have not been used for this purpose.

2.3.3 Metrics

To verify where and why a Web page is not accessible, it is important to analyse the HTML structure of the Web page. This analysis brings the possibility of measuring quantitatively the level of accessibility of a Web page. Further, metrics facilitate understanding, observation of experimental studies results and assessment of the results obtained.

The accessibility levels used by final users and public bodies are based in a WCAG 2.0 metric that use the results of the success criteria. There are five requirements that have to be accomplished in order for the content to be classified as conforming to WCAG 2.0 [19]. They are:

- *conformance level*: there are three levels of conformance – A (the minimum level of conformance), AA and AAA. The higher levels of conformance include the lower ones;
- *full pages*: conformance level is for the Web page as a whole, no part of a Web page can be excluded;
- *complete processes*: a Web page is part of a sequence of Web pages that need to be visited in order to accomplish an activity. All Web pages involved have to conform at the specified level or better;
- *only accessibility-supported ways of using technologies*: "Only accessibility-supported ways of using technologies are relied upon to satisfy the success criteria. Any information or functionality that is provided in a way that is not accessibility supported is also available in a way that is accessibility supported" [19];
- *non-interference*: technologies not accessibility supported can be used, if all the information is also available using technologies that are accessibility supported and as long as the non-accessibility-supported material does not interfere.

Several studies were conducted on quantifying Web accessibility in different ways. Some examples are:

- UWEM [35] defined a accessibility metric that result in a accessibility value for each Web page, using the *failure rate* for each Web page;
- all rates setted – *optimistic rate* and *conservative rate* and *strict rate* – were computed for checkpoint and aggregated into a final score of accessibility quality of a Web page [31];
- WAQM [37] computes the *failure rate* for each tested Web page and the average of the results for each Web page (considering the page weight in the site) is the final result.

Whereas these metrics provide different perspectives of the accessibility quality, none of them directly addresses the developers' effort in a way that relates with the common development process. Templates, as seen, are a fundamental part of this process and should be taken into account.

2.4 Existing tools

Web Accessibility Initiative (WAI) publishes a large list of Web Accessibility Evaluation Tools [1]. Some examples of those tools are:

- *A-Checker*[2] a free on-line accessibility checker that tests Web pages for conformance to WCAG 1.0 guidelines;
- *aDesigner*[6] a disability simulator of visual disabilities – low vision and blind people – that checks elements that may not be well properly used for people with these disabilities and it also checks accessibility WCAG 1.0 guidelines;
- *A-Prompt*[1] a free accessibility evaluator and repair tool that uses WCAG 1.0.

However, none of these tools uses WCAG 2.0 and most implementations of automatic evaluations do not consider the changes in the HTML document. Since experts' and users' evaluation are performed in the Web browser on the rendered state of the Web page, they do not suffer with these changes. To solve this problem, the accessibility evaluation should be applied to new execution environment, i.e., in the Web browser context - *Browser execution environment*. Consequently, evaluation can be performed *a priori* or *a posteriori*, i.e., before or after of the processing that happens in the Web browser, respectively.

The importance of the Web browser context in the evaluation results is starting to be considered and is already used in four tools [21]:

- *Foxability*[3] an accessibility analyzing extension for Firefox, that uses WCAG 1.0;
- *Mozilla/Firefox Accessibility Extension*[21] an extension of Firefox that uses WCAG 1.0 and perform report generation;
- *WAVE Firefox toolbar*[5] is a Web Accessibility Evaluation Tool that provides a mechanism for running WAVE reports directly within Firefox, using WCAG 1.0, but it does not perform reporting;
- *Hera-FXX Firefox extension* [21] semi-automatic tool to perform accessibility evaluation, using WCAG 1.0.

These tools focus only on use WCAG 1.0, which has been obsoleted by its latest 2.0 incarnation and they are embedded as extensions, becoming more limited in terms of their application. Because, they cannot be used outside the Web browser.

During the course of this work, Hera-FFX was updated [22] and its new version uses WCAG 2.0, but, this is only a semi-automatic tool and it cannot be used outside the Web

[1] WAI - Complete List of Web Accessibility Evaluation Tools: `http://www.w3.org/WAI/ER/tools/complete`

browser. Consequently, it does not allow comparison of evaluations of different execution environments. Up to this point, no work that focuses on the differences between results in different execution environments has been found. To perform correct comparisons, it must be guaranteed that tests are implemented in different execution environments in the same way, by reducing implementation bias.

2.5 Summary and Requirements

This chapter provided an overview of the main topics of WAEs, how they are necessary and their results. This way, it is possible to understand: the WAE standard used and why it was chosen, as the evaluation can be performed and under what circumstances, and the kind of problems that the developer has to solve, to repair accessibility errors. Besides, there were described the general problems of the existent WAE tools that performed the evaluation, showing the necessity of a new toll that performs this evaluation using more recent guidelines and in the rendered or transformed state of a Web page.

This chapter allowed understanding the state of the art and what still needs to be done in this area. This way, some important requirements are:

1. an automatic accessibility evaluator, which uses WCAG 2.0, has a test-bed to its validation and is able to perform the evaluation in the static and transformed HTML in a fair way to compare them;

2. a reporting mechanism with EARL, taking templates into account and accompanied by a fair development metric.

Chapter 3

Evaluation Framework

This chapter contains the specification of the accessibility evaluation framework. It presents its architecture, describing the several components needed, explains all the necessary design decisions, relatively to the components of the architecture, and detailed the implementation details and problems.

3.1 Architecture

In this work, two main execution environments were emphasized: *Command Line*, and *Browser*. In the *Command Line* execution environment, evaluation is performed on the HTML document that is transmitted initially in an HTTP response, whereas in the *Browser* execution environment, evaluation is targeted at the transformed version of the HTML document. To better grasp the differences between these execution environments, some requirements for the architecture of the evaluation framework were defined:

1. must be modular and flexible, allowing the addition of new components quickly and easily at any-time, without compromise the functionalities of the others components;

2. must allow a proven equivalence in both execution environments, so they can be compared fairly.

These requirements should be considered in the design of the framework's architecture. The architecture (depicted in Figure 3.1) is composed by four major components: the *QualWeb Evaluator*, *Execution Environments*, *Techniques* and *Formatters*.

The *QualWeb Evaluator* can be used to perform Web accessibility evaluations independently to the *Execution Environment* chosen. Because, the object of evaluation is a HTML DOM tree, which can be obtained at any moment.

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 [9], since EARL is a standard format for accessibility

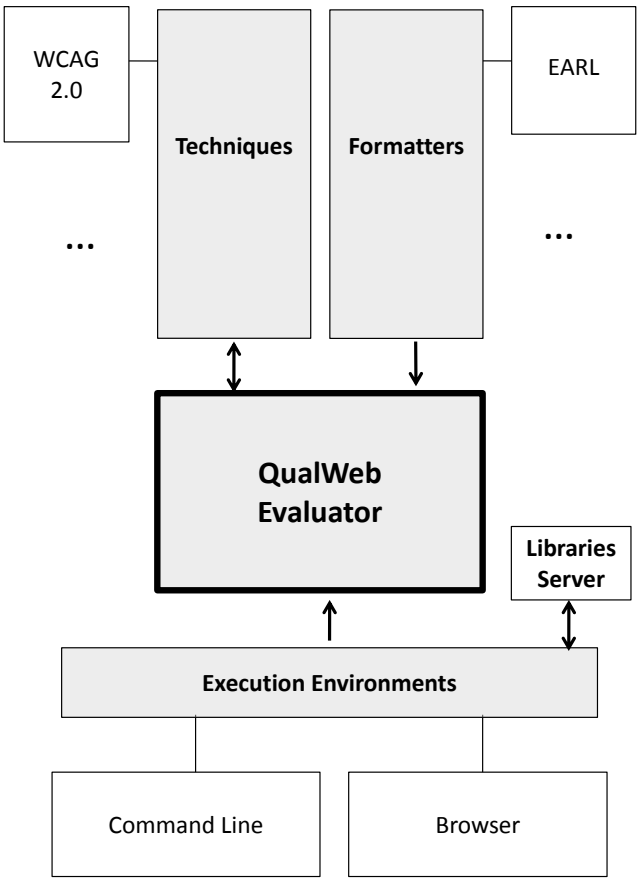

Figure 3.1: Architecture of the Evaluation Framework

reporting. This way, it can be interpreted by any tool that understands this format, and even allow comparing the results with other tools.

The *Execution Environments* component is responsible for the transformation of the Web page interpretation (HTML document) in an equivalent DOM representation. According with the state of the processing determined by the *Execution Environments*, i.e., if the Web page has already been processed or not.

The *QualWeb Evaluator* is composed by two components (Figure 3.2): *QualWeb Client* and *QualWeb Server*. The first one executes the evaluation on the data received from the *Execution Environments*, using the *Techniques module*. The second one is a Web server that receives evaluation results without the final serialization and serializes them, using the *Formatter module*, and, at the end, stores the final reports.

Finally, *Libraries Server* stores the libraries needed by the *Execution Environments*. Especially the libraries that have to be injected in the *Browser*'s environment.

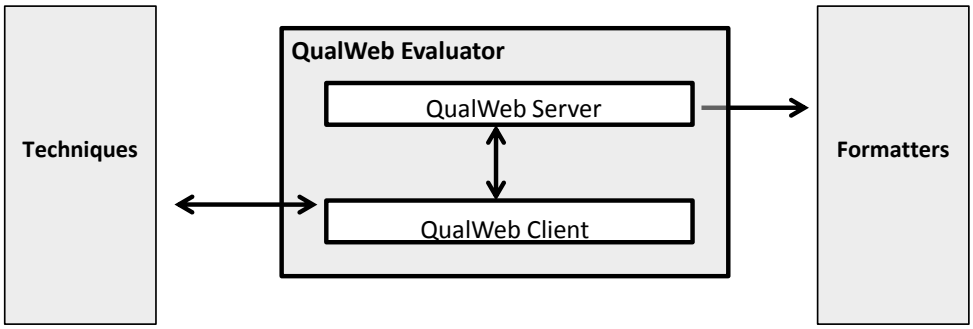

Figure 3.2: QualWeb evaluator sub-modules.

3.2 Execution Environments

To perform each assessment, both execution environments have different requirements and a different sequence of steps/actions. Figures 3.3 and 3.4 show the sequence in the *Command Line* and *Browser* execution environments, respectively. Consequently, the way of obtaining the HTML document has to be appropriated to the execution environment.

It was important to consider that some Web pages do automatic redirection for other URL, which can cause errors if the script used does not capture the HTML document for the redirected Web page. To mitigate this problem, it was used CURL [1], which is a command line tool that transfers data with URL syntax. With this toll the HTML document of the redirected Web page was obtained.

After the parsing of the HTML document into a HTML DOM tree, the *lowerCase* module was used. This module was developed in this work and it converts all the elements of a HTML DOM tree into lower case letters. This is done to prevent the occurrence of any possible error/problem during the search, which is case sensitive, since most Web sites contain a mixture of lower case and capital letters in its elements. Sequentially, the HTML DOM tree can be used by *QualWeb evaluator* and processed by the *Techniques* module, without problems.

Additionally, another module was necessary to determine the number of elements of a HTML DOM tree, and it is called *counter* module.

In the next section will be described the specific implementation of each execution environments.

[1] CURL: `http://curl.haxx.se/`

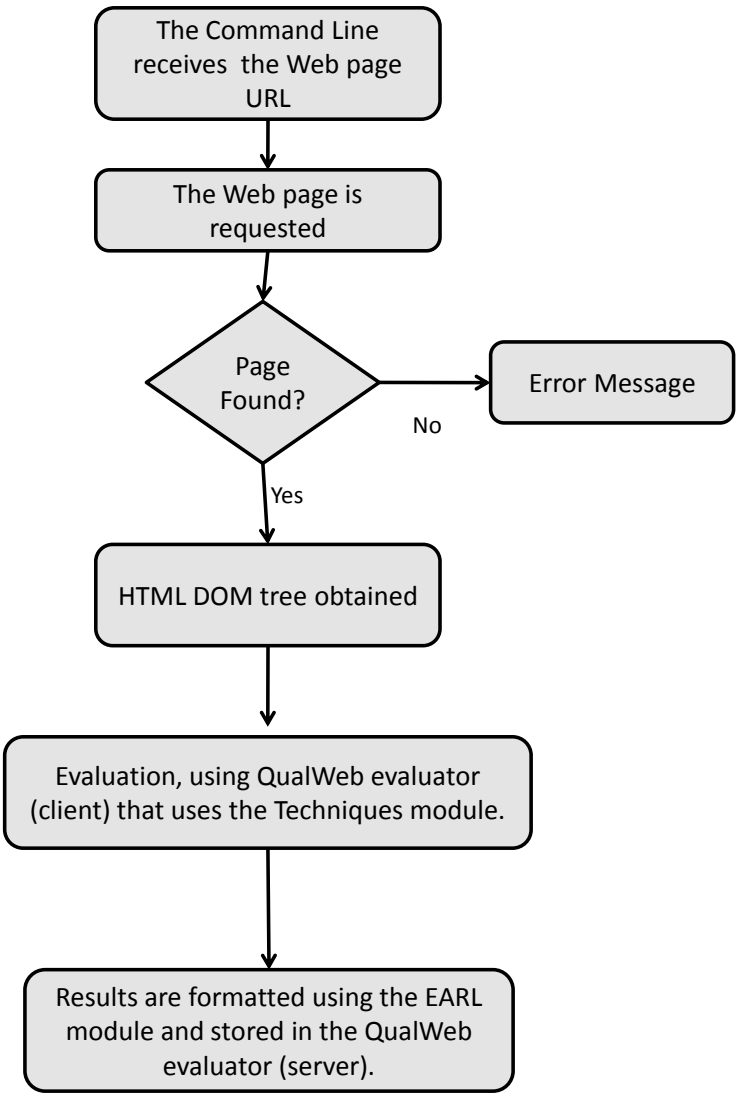

Figure 3.3: Flowchart of assessment in the Command Line execution environment.

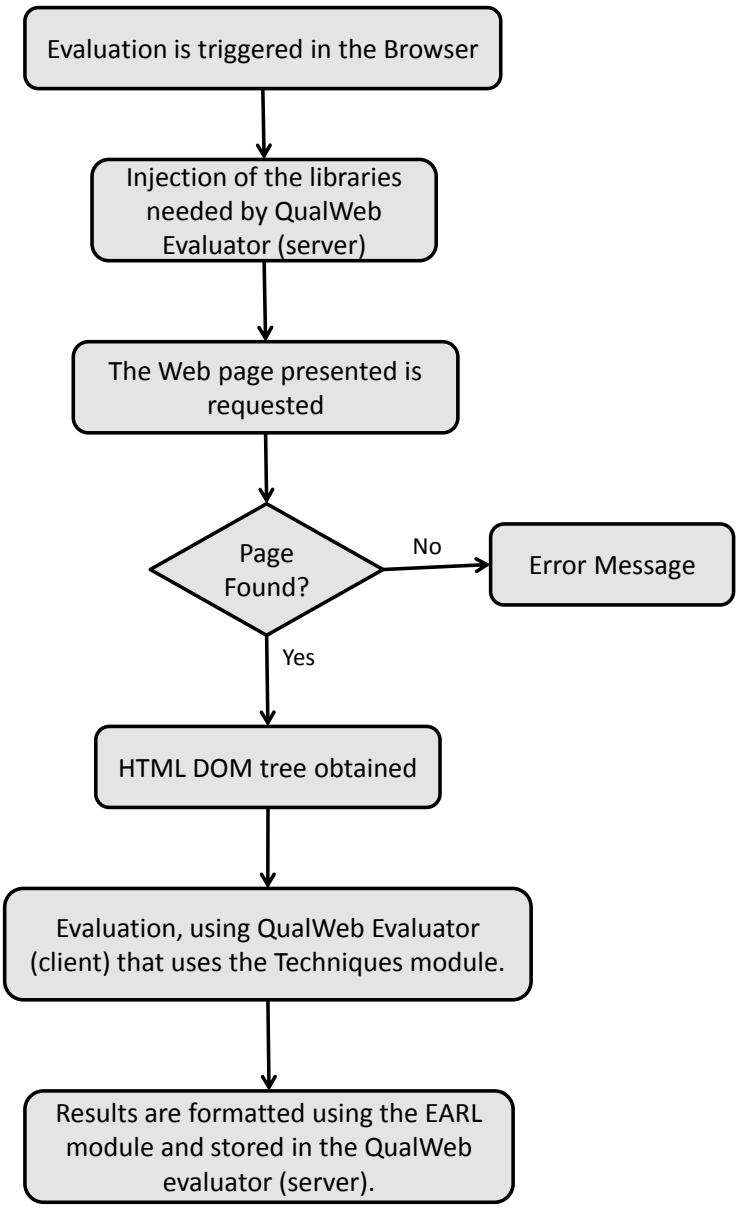

Figure 3.4: Flowchart of the sequence of assessment in the Browser execution environment

3.2.1 Command Line Environment

The sequence of execution and implementation of the evaluation process in this execution environment is the following:

1. a user provides the URL of a Web page;
2. the HTML document of the Web page is obtained, using a HTTP GET request;
3. if the HTML document is obtained, it is parsed in HTML DOM tree, using the *HTML Parser* module, and unified by the *lowerCase*;
4. all the libraries/modules needed (e.g., *Counter* module, *WCAG 2.0* module, *QualWeb evaluator* module) are available and the evaluation is performed on the HTML DOM tree, using *QualWeb evaluator*;
5. the evaluation results are sent to the *QualWeb evaluator Server*; and
6. the EARL serialization is performed and the data is stored.

3.2.2 Browser Environment

The sequence of execution and implementation of the evaluation process in this environment is the following:

1. a user triggers the evaluation in the Web browser, using a bookmarklet (Figure 3.5) (i.e., a line of Javascript stored on the URL of a bookmark, to trigger the execution of the evaluation within the Web browser). When the user activates the bookmarklet, these commands are run. To implement this *browser-server* execution and communication mechanism, the following modules were used:
   (a) *Bootstrap*, to import several of the required base modules. This way, only one line of Javascript, which executes a bootstrap file that imports all the libraries/modules needed, has to be written. This was necessary, because the code stored in the *bookmarklet* has a limited number of characters (that depends on the Web browser used); and
   (b) *LAB.js*, to inject all the evaluation modules into the browser's DOM context.

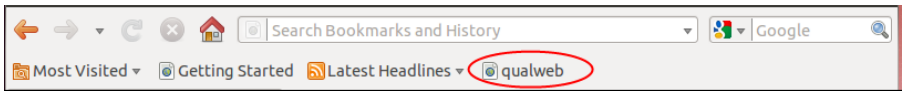

Figure 3.5: Evaluation execution example on Browser

2. the bookmarklet injects the required modules, contained in the *Libraries Server*, to: obtain the HTML DOM tree of the current Web page, using the Node-HTMLParser; execute the evaluation, using the *WCAG 2.0* module and *QualWeb evaluator* module); and to count the elements, using the *Counter* module;

3. the evaluation is performed on the HTML DOM tree;

4. after the evaluation is performed, to allow the results to be accessed outside the Browser, a *form* with a propriety *visible* with the value false, is injected into the Web page. That *form* contains an element *textarea* that is filled with the evaluation results;

5. the results are sent to the *QualWeb evaluator Server*, using a HTTP POST of the *form*; and

6. the EARL serialization is performed and data is stored.

To obtain the HTML document of a Web page in the Browser, the following methods were considered:

- *document.Head.innerHTML* + *document.Body.innerHTML* to obtain the head and the body of the page. However, this method could not obtain the entire HTML document;

- *document.documentElement.innerHTML* to obtain all the HTML of the Web page. However, it had the same result as the previous method;

- finally, the function presented in Figure 3.6, which was able to capture all the HTML document of the Web page, including the HTML element (i.e., HTML tag).

```
var outerHTML = function (node){
        return node.outerHTML ||
        new XMLSerializer().serializeToString(node);
}
```

Figure 3.6: Function to obtain the HTML document of the presented Web page.

3.3 QualWeb Evaluator

The *QualWeb evaluator* is divided in two sub-modules. Because of that, the implementation of this component will be explained separately in *QualWeb evaluator Client* and *QualWeb evaluator Server*.

3.3.1 QualWeb Evaluator Client

The *QualWeb evaluator* Client receives the HTML DOM tree of a Web page from an execution environment. Sequentially, it performs the WAE, using all the techniques available in the *Techniques* component chosen. This way, an *array of results* is created (i.e., an array of objects that contains all the accessibility results from HTML DOM tree). Each element of the array is composed by the *position* of the element evaluated in the HTML DOM tree, the *technique* used to evaluate the element and the *result* of its evaluation (Figure 3.7).

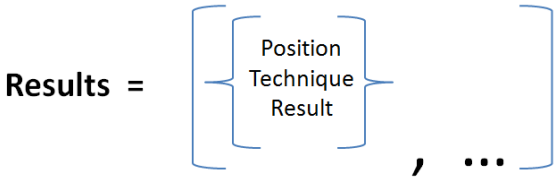

Figure 3.7: Scheme of the array of results.

Besides, a metadata object needed to be introduced into the results, to support the specification of the *elements count* and a *timestamp*. The *elements count* indicate the total number of elements in a Web page, because some elements do not have applicability and they do not present any accessibility result/outcome. The *timestamp* states the specific time when the evaluation was performed, to allow the comparison of evaluation times. Consequently, final results are represented by joining the *array of results* shown in Figure 3.7 with the *metadata*, as shown in Figure 3.8.

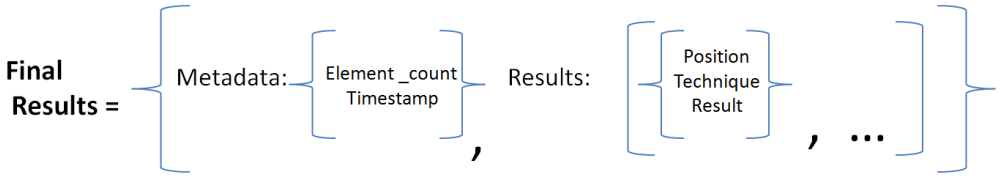

Figure 3.8: Scheme of the new representation of the results.

After the execution of the Web Accessibility Evaluation, the results are serialized, using a type of *Formatters* component.

3.3.2 QualWeb Evaluator Server

The *QualWeb evaluator Server* has two functions: (1) it allows access to certain libraries required in the Browser's execution environment, and (2) it receives the evaluation results,

and performs the serialization. For the first task, it was used the *Node-Static* module and, for the second task, it was used the *Node-Router* module.

When the results are received, all the headers from the HTTP POST were removed. The results were transformed in the EARL serialisation format, using the EARL module, and subsequently stored in this component (*QualWeb evaluator Server*).

3.4 Techniques

In this component, it is possible to select what techniques/technique to use.

For each technique, the HTML DOM tree of the Web page is traverse until needed, to verify if the critical elements of the HTML DOM tree considered in the technique were in accordance with the specific techniques recommendations. The search for elements, with potential problems, is done in the most efficient way. Consequently, it does not examine the unnecessary elements, the elements that could not have the critical problems that are being searched.

Finally, depending on the results of that accordance, a specific value of outcome is assigned for the element, and for each of these results a new element is added to the *array of the results*.

3.4.1 WCAG 2.0

There is a total of 54 HTML WCAG 2.0 Techniques, but only 18 HTML techniques were implemented. Table 3.1 presents the chosen techniques and their descriptions. The techniques take into account the criteria showed in Table 3.2.

Other techniques were not chosen, because most of them require a complex processing of semantics, sound, video, and other media, which was not an objective of this thesis. In fact, to the best of our knowledge, there is no Web accessibility evaluator that even addresses the issue and the known media processing techniques are not accurate enough to offer unambiguous classifications. Therefore, if techniques that need this kind of processing were chosen, using the existent media processing tools, they could lead to incorrect results.

Figure 3.9 shows an excerpt from WCAG 2.0 H64 technique. It can be observed a search in the HTML DOM tree of frames or iframes elements, to determine whether or not they have a title.

Table 3.1: Techniques Implemented

Techniques	Description
H2	Combining adjacent image and text links for the same resource
H25	Providing a title using the title element
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
H36	Using alt attributes on images used as submit buttons
H37	Using alt attributes on img elements
H44	Using label elements to associate text labels with form controls
H46	Using noembed with embed
H53	Using the body of the object element
H57	Using language attributes on the html element
H64	Using the title attribute of the frame and iframe elements
H65	Using the title attribute to identify form controls when the label element cannot be used
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
H93	Ensuring that id attributes are unique on a Web page

Table 3.2: Criteria Considered

Criterion	Description
1.1.1	Non-text Content that is presented to the user has a text alternative, except for pre-defined situations
1.2.3	Audio Description or Media Alternative content is provided for synchronized media, except when the media is a media alternative for text and is clearly labelled as such
1.2.8	Media Alternative is provided for all pre-recorded synchronized media and for all pre-recorded video-only media
1.3.1	Info and Relationships conveyed through presentation can be determined or are available in text
2.4.1	A mechanism is available to bypass blocks of content that are repeated on multiple Web pages
2.4.2	Web pages have titles that describe topic or purpose
2.4.4	Link Purpose can be determined from the link text alone or from the link text together with its determined link context, except where the purpose of the link would be ambiguous to users in general
2.4.9	Link Purpose can be identified from link text alone, except where the purpose of the link would be ambiguous to users in general
3.1.1	Language of Page can be determined
3.2.2	Changing the setting of any user interface component does not automatically cause a change of context unless the user has been advised of the behaviour before using the component
3.2.5	Changes on request of user
3.3.2	Labels or instructions are provided when content requires user input
3.3.5	Help is available
4.1.1	Elements have complete start and end tags, elements are nested according to their specifications, elements do not contain duplicate attributes, and any IDs are unique, except where the specifications allow these features
4.1.2	For all user interface components, the name and role can be determined; and values that can be set by the user can be set

```
function inspect(DOMList)
{
    if (typeof DOMList == "undefined" || DOMList.length == 0)
        return;

    for (var i = 0; i < DOMList.length; i++)
    {
        position++;
            if (DOMList[i]["type"] == "tag" && (DOMList[i]["name"]
            == "frame" || DOMList[i]["name"] == "iframe"))
            {
                    if(DOMList[i]["attribs"]["title"] != "" &&
                    DOMList[i]["attribs"]["title"] != "undefined")
                    {
                        addElement(position,'warning',"");
                    }
                    else
                      addElement(position,'failed',"");
                  }
                  inspect(DOMList[i]["children"]);
          }
}
```

Figure 3.9: Excerpt from WCAG 2.0 H64 technique

3.5 Formatters

This component receives the *final results*, which are used to produced its chosen serialization, according with the type of *Formatters* selected. The *Formatter* used will serve as the basis for generating CSV reports that will allow statistical analysis of the results.

In the case of this work, as mentioned before, EARL serialization was chosen and it will be detailed in the next sub-section.

3.5.1 EARL

For this type of reports generation, schemes/templates of the reports were defined, using the *Node-Template* module. These schemes/templates contain the generic information needed for reporting. The more specific content will be generated by applying the evaluation results to those schemes/templates to construct the EARL reports.

Figure 3.10 shows a simple example of the use of the *Node-Template* module. The example shows the creation of a Test Mode XML class that describes how a test was carried out for the assertion with number *index* (that refers to one of the results), in this case the test was automatic.

```
var testm =
'<earl:Assertion rdf:about="#assertion<%=index%>">'+
'<earl:mode rdf:resource="... /earl#automatic"/>'+
'</earl:Assertion>';

var temp = template.create(testm);

earl += temp({index:index});
```

Figure 3.10: Example of Node-Template module application.

EARL's original specification does not support the introduction of new data. For that reason, it had to be extended, to include another small set of elements, introduced as *metadata* (mention in section 3.3.1). For that, a *metadata* XML class was defined to add extra information when needed. In this case, the *metadata* class supports the specification of *element count*, and *timestamp*. However, other elements can be added, if necessary. Because of the *metadata* class, the results received by any *Formatter* module have to contain the information necessary to fill these fields. Figure 3.11 shows an EARL document example in RDF/N3[2] format.

Relatively to the CSV reports generation, a *EARL-Parser* module was used to parse the EARL, and *CSV* module was used to allow a better inspection and statistical analysis with off-the-shelf spreadsheet software. Due to the extensiveness of EARL reports that

[2]RDF/N3:RDF/N3:http://www.w3.org/DesignIssues/Notation3

```
<#QualWeb> dct:description ""@en;
        dct:hasVersion "0.1";
        dct:location "http://qualweb.di.fc.ul.pt/";
        dct:title "The QualWeb WCAG 2.0 evaluator"@en;
        a earl:Software.
<assertion1> dc:date "1291630729208";
        a earl:Assertion;
        earl:assertedBy <assertor>;
        earl:mode earl:automatic;
        earl:result <result1>;
        earl:subject <http://ameblo.jp/>;
        earl:test <http://www.w3.org/TR/WCAG20-TECHS/H25#H25>.
<http://ameblo.jp/> dct:description ""@en;
        dct:title "The QualWeb WCAG 2.0 evaluator"@en;
        qw:elementCount "381";
        a qw:metadata,
        earl:TestSubject.
<http://www.w3.org/TR/WCAG20-TECHS/H25> dct:hasPart
<http://www.w3.org/TR/WCAG20-TECHS/H25#H25-tests/>;
        dct:isPartOf <http://www.w3.org/TR/WCAG20-TECHS/>;
        dct:title "H25"@en;
        a earl:TestCase.
<QualWeb> dct:description ""@en;
        dct:hasVersion "0.1";
        dct:title "The QualWeb WCAG 2.0 evaluator"@en;
        a earl:Software;
        foaf:homepage qw:.
<result1> dct:description "description"^^rdf:XMLLiteral;
        dct:title "Markup Valid"@en;
        a earl:TestResult;
        earl:info "info"^^rdf:XMLLiteral;
        earl:outcome earl:passed;
        earl:pointer <1>.
```

Figure 3.11: EARL document

can be generated by the evaluator (sometimes with thousands of lines), the EARL-CSV transformation had to be performed using a SAX parser. Because, generic DOM parsers would be significantly slower and they would cause a significant memory consumption.

3.6 Template-based Evaluation

It is important to perform correct reporting, taking into account that the same accessibility errors can be repeated many times in the Web sites/pages, as previous mentioned. Because of that, it is necessary to perform a template-based evaluation.

This evaluation can be performed in the Web page source or after the Web page processing, i.e., a priori or a posteriori. However, in the first approach two accessibility evaluations and two distinct periods of correction would be necessary, because some accessibility problems could be introduced by the Web page processing. Consequently, it was decided to perform the template-based evaluation into the transformed HTML (final stage of the page).

The *Fast Match* algorithm [14] (detailed in the next sub-section) was selected to perform Web page template detection. Because of its applicability and adequacy to search similarities between two node trees. However, the Web pages are transformed in their representation in a HTML DOM tree, so that the similarities can be detected by the *Fast Match* algorithm, allowing the identification of the common elements between two Web pages.

Another advantage of this algorithm is its execution time. This is important, because Web sites can have many Web pages, which can be represented with large trees. Consequently, the algorithm has to be fast.

The application of this algorithm to a Web site or between Web sites will result in a precise enough measure of the components that are part of a template, or should probably be part of one.

Before the algorithm is executed, two other functions have to be use: 1) a function that assures that every element has a *unique identifier*, creating a new one if a element has none, or replacing it in case there is any repeated; and 2) a function that parses the HTML DOM tree making the correspondences between the *unique identifier* of an element and its element number. The first one is necessary, because the *Fast Mast* algorithm generates pars of commons elements, using their *identifiers*. The second one is necessary to simplify how to find an element that has a match in a HTML DOM tree, i.e., the element can be directly accessed.

Finally, the execution of the *Fast Match* and the detection of the Web page templates detected will lead to the division of the accessibility evaluation results in two sets: *template* set and *specific* set. The first set contains the evaluation of the common parts between two Web pages. The second set contains the evaluation of different HTML elements

of the Web pages, i.e., part of the *specific* structure of a Web page. This division will allow a faster access to the type of accessibility evaluation results desired.

3.6.1 Fast Match algorithm

It is proposed the use of this algorithm to identify common elements amongst the HTML DOM trees. Although this algorithm will only provide an approximation of the template elements used in its construction, it will offer a reasonable estimate for initial assessment. On the other hand, it will also raise the developers' awareness to other common elements, not contained in templates that could be addressed in the corrective processes and that should be added to the templates to improve the process of code reuse.

To detect a template that is present in all the Web site pages considered, the matches obtained for each Web page have to be compared. However, in this work, the template detection technique developed can only be used in two Web pages to detect templates. Because, the objective is to verify if template detection is really advantageous to accessibility proposes. Consequently, if this happens, this algorithm will be upgraded in order to compare all the Web pages of a site.

The *Fast Match* algorithm required of the definition of a few variables: a *label*, which is an HTML element name, a *value* that is a data text of the nodes, and a *unique identifier* that represent a HTML element. To check whether or not two elements match, it was necessary to define a *compare* function. This function is based on the Levenshtein Distance [33], which measures the amount of difference between two *values*.

Another two thresholds, for the *Fast Match* algorithm, were defined: t and f. These are needed to control the range of valid results of two functions used in the algorithm. The first one – t – is the ratio of equal descendent elements between two elements for these to be considered equals. The second one – f – is the maximum value that limits the *compare* function's result. Some tests were performed to define reliable thresholds values, i.e., it was applied the *compare* function and the ratio of equal descendants between two nodes and observe the frequency of the results to define the thresholds. However, the frequency analysis was inconclusive and, because of that, several values have to be tried, within the ranges of f (between 0 an 1) and t (bigger or equal than 0.5). The observations yielded $t = 0.5$ and $f = 0.5$ as optimal values for HTML DOM trees.

For the tests carried out, to determine and adjust these valued, 7 Web sites were used. The selection rationale was to select well-known and representative Web sites from the Alexa Top 100 Web sites[3] – *Google*,*Wikipedia*, *Facebook* and *Amazon* –, two modern online Portuguese newspapers – *DN* and *Público* – and the *WordTaps*. *WordTaps* uses *WordPress*, which is a well-known blogging and Web site platform.

To perform the matches, new fields were inserted to mark the elements as *matched*. This way, in the beginning, all the elements should be marked as *unmatched* (*matched*

[3] Alexa Top 100: http://www.alexa.com/topsites

= 0) and when a match is found the elements should be marked as *matched* (*matched* = 1). Consequently, if an element has a match, the algorithm does not try to find it another match.

The algorithm [14] receives two HTML DOM trees, T_1 and T_2, uses two auxiliary arrays S_1 and S_2 to store data, and is executed as follows:

1. a match's array is created, *M*;
2. For each leaf with a label *l* do
   (a) all the content in S_1 and S_2 is deleted;
   (b) determine all of the nodes with label *l*, for T_1 and stores that in S_1
   (c) determine all of the nodes with label *l*, for T_2 and stores that in S_2
   (d) Longest Common Subsequence (*LCS*) between S_1 and S_2 is performed. $LCS(S_1, S_2) = (x_1, y_1)...(x_n, y_n)$; and $x_1...x_n$ is a subsequence of S_1; $y_1...y_n$ is a subsequence of S_2; and $equal(x_i, y_i)$ is true for $1 \leq i \leq k$, and
      i. for leaf nodes: $equal(x_i, y_i)$ is true, $l(x) = l(y)$ and $compare(value(x), value(y)) \leq f$
      ii. for other nodes: if the ratio of equal descendent elements between two elements $\geq t$
   (e) all the *LCS* pairs are added into *M*, and the nodes are marked as *matched*;
   (f) for each unmatched node $x \in S_1$, if there is an *unmatched* node $y \in S_1$, such that $equal(x, y)$. They should be added into *M*, and marked as *matched*
3. the steps 2-2f should be repeated for each internal node label *l*

Finally, after all the parameters and functions were defined, the matches obtained with the algorithm were compared and verified with the same matches conceived manually. This was important, to assure that the results obtained were correct.

3.6.2 A Template-Aware Web Accessibility Metric

The Template-based Evaluation for each Web page of a Web site allows to set a new metric of accessibly, taking into account the applicability in Web page templates and in the specific part of a Web page. Using the applicability results for each page it can be composed the applicability for the entire Web site. The metric defined was:

$$\alpha(p_i) = \begin{cases} \alpha_t \\ \alpha_s(p_i) \end{cases} \tag{3.1}$$

Equation (3.1) indicates the applicability of a Web page – $\alpha(p_i)$ – , containing the absolute value or percentage of nodes in the template applicability – α_t – and the absolute value or percentage of nodes in specific applicability – $\alpha_s(P_i)$.

$$\alpha(S) = \begin{cases} \alpha_t \\ \alpha_s(p_i), \forall\, p_i \in\ S \end{cases} \tag{3.2}$$

Equation (3.2) indicates the applicability of a Web site – $\alpha(S)$ – and it contains the template applicability – α_t – as the first one, and the absolute value or percentage of nodes in specific applicability – $\alpha_s(p_i), \forall\, p_i \in S$ – for each page of the Web site.

Based on the quantity of templates used, presented in the metric, it is a more correct measure of the effort to the error reparation. This applicability metric can be used by other accessibility metrics, improving and making them more real and helpful.

In the next sub-section, it will be detailed some implementation details.

3.7 Implementation details

In order to compare the proposed execution environments, the same implementation for the accessibility evaluation had to be used. Given that, one of the execution environments is the Web browser, creating a restriction on using Javascript as the implementation language. Thus, to develop the *Command Line* version of the evaluation process, it was used *Node.js*[4], an event I/O framework based on the V8 Javascript engine[5].

In addition to standard *Node.js* modules, several other ancillary modules were used[6], including:

- *Node-Static* allows serving static files into the Browser execution environment;
- *Node-Router* supports the development of dynamic behaviours, which was used to implement the retrieval and processing of evaluation result;
- *Node-Template* allows using pre-defined templates for each XML classe that compose the EARL files;
- *Libxmljs* parses EARL reports using a SAX parser,
- *Node-HTMLParser* provides support for building HTML DOM trees in Browser execution environment, and
- *HTML-Parser* provides support for building HTML DOM trees in Command Line execution environment.

[4]Node.js: `http://nodejs.org`
[5]V8 Javascript engine: `http://code.google.com/p/v8/`
[6]GitHub modules: `https://github.com/ry/node/wiki/modules`

Besides these, a set of modules/libraries were implemented, for the evaluation framework, including:

- *Qualweb Evaluator module*, which performs the accessibility evaluation with the implemented techniques;
- *WCAG 2.0 Techniques module*, which contains all the guidelines and criteria implemented;
- *EARL module*, which allows for the creation of EARL documents with the defined templates;
- *EARL-Parser*, which parse EARL files using the *Libxmljs* library; and
- *CSV* module, to recreate a *comma-separated-values* (CSV) counterpart from a given EARL report;
- *Template-detection* module, which performs the detection of the common elements;
- *lowerCase* module, which converts all the elements of a HTML DOM tree into lower case letters;
- *counter* module, which determines the number of elements of a HTML DOM tree.

The previous lists resume all the modules/libraries used.

3.8 Summary

In this chapter, it was defined the architecture of the evaluation framework, its requirements and all that was necessary to develop for its implementation. More specifically, the techniques that had to be implemented, the match algorithm used, the template-based evaluation and a new accessibility metric.

Finally, there were also described the various components of this Evaluation Framework, presented the various libraries used and some specific implementations details.

Chapter 4

Evaluation

In this chapter will be detailed the validation of the techniques implemented and two experimental studies the were performed. Each experimental studies has a research hypothesis and goals:

- the first experimental study aims to verify if *Web content in the Web browser provides more accurate and more in-depth analysis of its accessibility – H1 –*, and the goals are:
  - understand the differences in the HTML between execution environments;
  - discover the limitations of accessibility evaluation in different execution environments;
  - assuring that the evaluation procedures are the same in all execution environments, so that they can be compared;
- the second experimental study aims to verify if *template-awareness simplify assessment reporting* – H2 – and the goals are:
  - provide an approximation of the templates' elements in its construction and a reasonable estimate for initial assessments;
  - apply the template-aware metric in a few Web sites.

4.1 Validation of WCAG 2.0 Techniques Implementation

A test-bed was developed, in order to verify that all the WCAG 2.0 implemented techniques provide the expected results. Therefore, it can be guaranteed that the evaluation outcomes are applied correctly, reports are corrected and thus metrics that can be applied also give the correct results, i.e., the framework quality can be assured.

The test-bed is constituted by a set of HTML test documents. The HTML test documents should be based on documented WCAG 2.0 techniques and ancillary WCAG 2.0

documents. Besides, each HTML test document should be 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 all the possible techniques' outcomes. To get a better perspective on the implementation of the tests, the examples of success or failure cases described, for each WCAG 2.0 technique used.

All the success or failure cases described for each WCAG 2.0 technique were considered and it was developed a total of 102 HTML documents. In the chart of Figure 4.1, it is presented the number of HTML test documents defined for each technique that will be implemented for the *QualWeb evaluator*. The number of HTML test documents for each technique depends on the number of fail, pass or warning cases possible. Finally, to ensure that the evaluation outcomes are not modified when changing execution environments, the same HTML test documents have to be used in both execution environments.

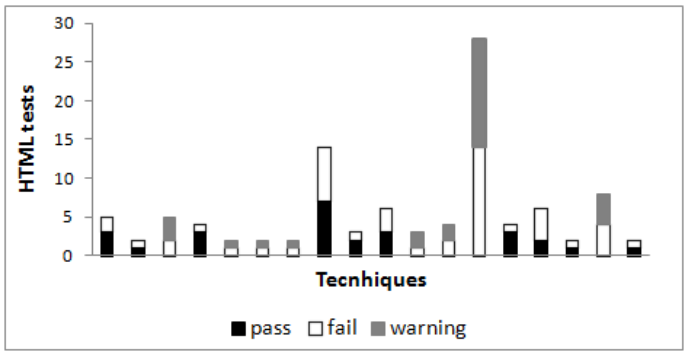

Figure 4.1: Number of Test Documents per Technique

The HTML test files were as simple as possible and they were focused on what is intended to verify in the techniques. Figures 4.2 and 4.3 show examples of HTML documents of the test-bed of the WCAG 2.0 techniques. Figure 4.2 shows an example of the correct application of the technique H25, i.e., the HTML document does have a title. In opposite, Figure 4.3 shows an example of the wrong application of technique H25, i.e., the HTML document does not have a title.

After the implementation of the HTML test documents, a small meta-evaluation of the techniques was performed, to guarantee its proper application. This meta-evaluation consisted in triggering the evaluation of the *Command Line* with a small automation script, as well as opening each of the HTML test documents in the *Browser*, and triggering the evaluation. All techniques were tested with the test-bed and a few implementation errors were detected and corrected, which was the objective of the development of the test-bed.

Afterwards, the evaluation outcomes (warn/pass/fail by technique) for all HTML test documents were compared with the previously defined expected results. Since all of these HTML tests documents will not include Javascript-based dynamics that transform their respective HTML DOM tree, it was postulated that the implementation will return the

```
<html xmlns= "http://www.w3.org/1999/xhtml">
        <head>
                <title>Title</title>
        </head>
        <body>
                <h1> ola</h1>
        </body>
</html>
```

Figure 4.2: A HTML test document with an example of the right application of technique H25.

```
<html xmlns= "http://www.w3.org/1999/xhtml">
        <head>
        </head>
        <body>
                <h1> ola</h1>
        </body>
</html>
```

Figure 4.3: A HTML test document with an example of the wrong application of technique H25.

same evaluation results in both execution environments.

4.2 Experimental Study 1 - Web Accessibility Evaluation

This experimental study was performed on the home pages from the Alexa's Top 100 Web sites[1]. It is centred on analysing how Web accessibility evaluation results in different outcomes for the *Command Line* and *Browser* execution environments.

In the next section will be detailed the setup of this experiment, followed by a description of how data was acquired and processed. Finally, it will be presented the most significant results of the experiment.

4.2.1 Setup

Initially, it was verified, for each Web site, if it could be reached, and if a valid HTTP response was obtainable, with the Web site's corresponding home page. For those which passed this verification, some of them had to be ignored. In one of them, the domain is used for serving ancillary resources for other Web sites. Others were filtered, since they are blocked by the university's network (mostly illegal file sharing or adult content

[1]Alexa Top 100: `http://http://www.alexa.com/topsites`

services). Finally, due to unknown reasons, some Web sites were unavailable, and had to be ignored for this setup.

The resulting set of Web sites that were to be evaluated comprises a total of 82 reachable home pages.

4.2.2 Data Acquisition and Processing

Both the original HTML documents (through the *Command Line* execution environment) and the transformed HTML documents (through the *Browser* execution environment) of the accessed Web pages were saved, so the assessments of these documents could be repeated, if necessary.

The evaluations were performed in both execution environments sequentially to the same Web page, and with little temporal differences. This way, the potential content differences between the HTTP responses, in both execution environments, were avoided, preventing incorrect evaluation results. The resulting time delta between evaluations of both execution environments averages at 89.72 seconds, $\sigma = 69.59$.

In some cases, on the Browser execution environment, strong safeguards were faced that deflected the ability to inject evaluation procedures into the HTML document (often implemented as safeguards for cross-site scripting attacks). For these cases, the restrictions were eliminated and the documents were successfully evaluated afterwards.

Finally, with all the evaluations finished, all EARL results were transformed into the corresponding CSV format for subsequent analysis, as mentioned in a previous section.

The evaluation yielded differences in the size of the HTML documents, both in terms of absolute bytes and HTML elements, when comparing these numbers between execution environments. The average difference on the byte size of the documents is 2885 bytes, $\sigma = 51181.63$, which supports the idea that Web pages can have several transformations in their content between execution environments. In terms of HTML element count, there is an average difference of 72.5 elements, $\sigma = 693.56$. These results indicate that, in fact, there are differences in the HTML between these two execution environments.

These numbers were further investigated, in order to understand if any of the cases, where the size of the documents, in bytes and number of HTML elements, increase or decrease in absolute values. These results are depicted in Figures 4.4 and 4.5, respectively.

In terms of absolute byte size for the evaluated Web pages, the *Command Line* execution environment yields an average of 69794 bytes, $\sigma = 95358.67$, while averaging at 81007.02 bytes in the *Browser* execution environment, $\sigma = 126847.75$. This scenario repeats itself for HTML elements, where the *Command Line* clocks at 915.71 elements on average, $\sigma = 1152.11$, and 1154.72 elements on average for the *Browser* execution environment, $\sigma = 1565.87$.

This outcome reflects the underlaying assumption made in the hypothesis, i.e., that the difference between HTML documents in both execution environments is real, and

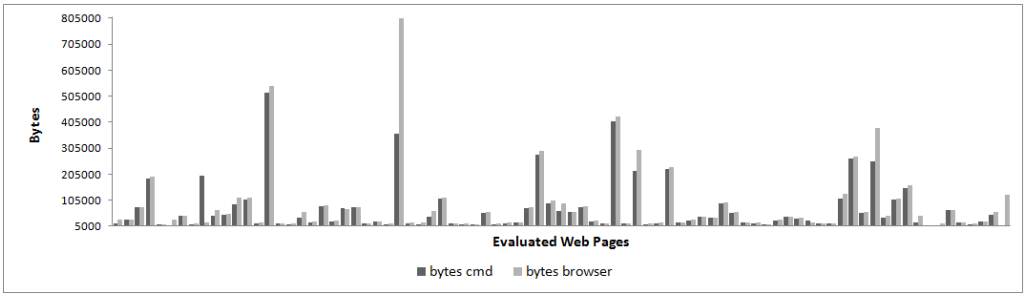

Figure 4.4: Comparing size in bytes in both execution environments

very significative. Based on this, it will be presented, in the next section, an analysis on how accessibility evaluation – based on WCAG 2.0 – becomes evident on the *Command Line* and *Browser* execution environments.

4.2.3 Results

The study was focused in two main set of results: first, the difference of evaluation outcomes (*fail*, *pass*, *warning*) between both execution environments; and second, what WAE criteria are able to characterise the differences between evaluating in each execution environment. The next sections detail the corresponding findings.

Evaluation Outcomes

It was detected that there are significant differences in the number of HTML elements detected by WAE procedures between both execution environments. In Figures 4.6, 4.7, and 4.8 it is presented how the three evaluation outcomes (*fail*, *pass*, *warn*, respectively) differ between execution environments. A *failure* occurs in cases where the evaluator can automatically and unambiguously detect if a given HTML element has an accessibility problem, whereas the *passing* represents its opposite. *Warnings* are raised when the evaluator can partially detect accessibility problems, but which might require additional inspection (often by experts).

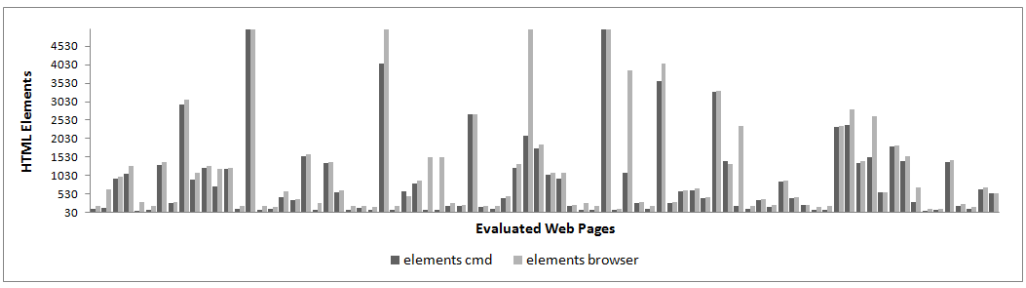

Figure 4.5: Comparing size in HTML Elements count in both execution environments

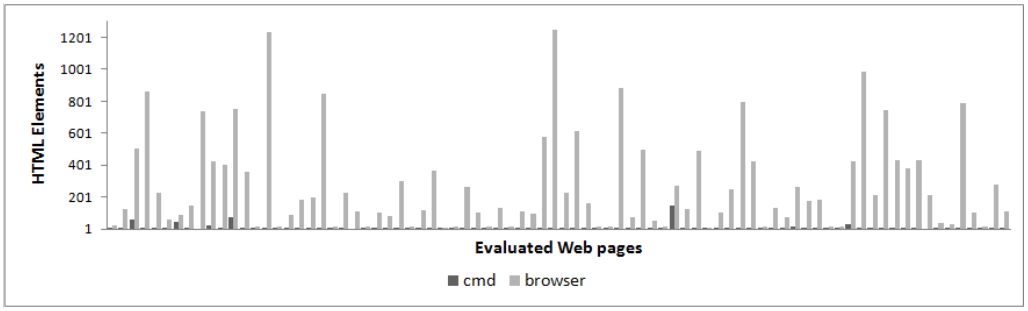

Figure 4.6: Number of HTML Elements that Passed

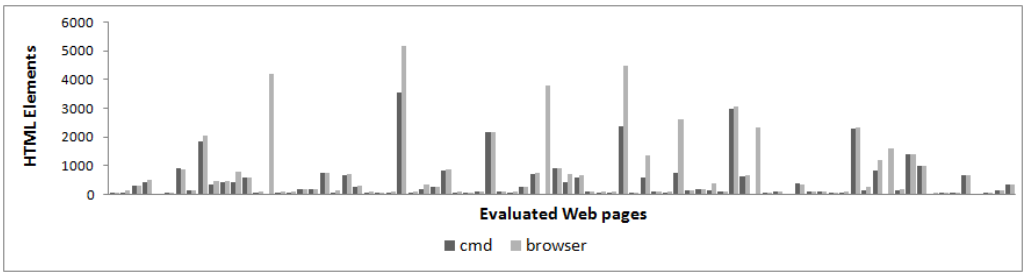

Figure 4.7: Number of HTML Elements that Failed

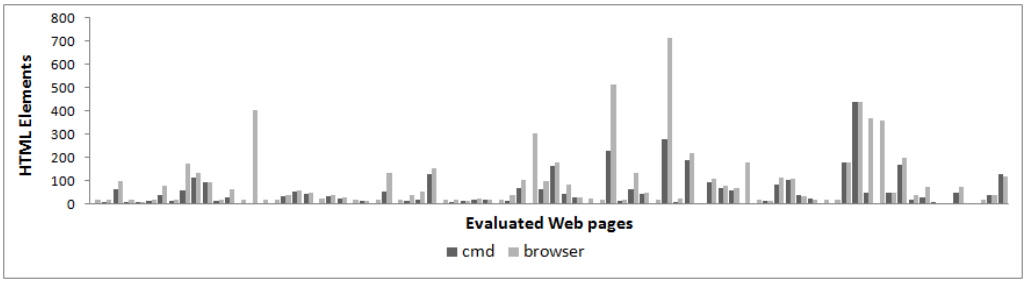

Figure 4.8: Number of HTML Elements that had Warnings

Inspecting these results with additional detail, the Web pages have the following evaluation outcomes:

- *Pass*: an average of 9.67 elements pass their respective evaluation criteria ($\sigma = 19.12$) in the Command Line execution environment. However, this number highly increases in the Browser execution environment to an average of 272.78 elements ($\sigma = 297.10$);
- *Fail*: an average of 47.44 elements fail their respective evaluation criteria ($\sigma = 70.82$) in the Command Line execution environment. This number increases in the Browser execution environment to an average of 90.10 elements ($\sigma = 125.93$);
- *Warn*: an average of 425.02 elements produce warnings in their respective evaluation criteria ($\sigma = 682.53$) in the Command Line execution environment. This number increases in the Browser execution environment to an average of 685.21 elements ($\sigma = 1078.10$).

In the next section will be described in detail how evaluation criteria differentiate between both execution environments.

Evaluation Criteria

WCAG 2.0 defines a set of evaluation criteria for each of its general accessibility guidelines. This experimental study resulted in several interesting outcomes from the accessibility evaluation. As it can be grasped from Figure 4.9 (log-scale on HTML Elements count), each implemented criteria is invariantly applied more times in the Browser execution environment than in the Command Line execution environment.

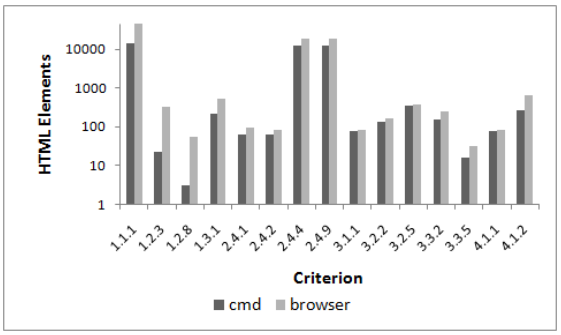

Figure 4.9: Browser vs Command Line per criterion (log-scale on HTML Elements count)

However, these results still mask an important detail about criterion applicability: there might be Web pages where any given criterion could be applied in the Command Line execution environment, but dismissed in the Browser execution environment (i.e., *false positives*). Likewise, the opposite situation can also occur (i.e., *false negatives*). In

other words, *false negatives* and *false positives* occur due to the differences between evaluation results of both execution environments, for instance, failing on Criterion 1.1 (i.e., alternative texts) in Command Line evaluation, but passing in the Browser (e.g., a script introduced alternative texts for images). This is a false negative yield by Command Line evaluation, since users are faced with its Browser counterpart.

Consequently, in this analysis, some cases were discovered where specific criteria in fact resulted in both *false positives* and *false negatives*, when using the Command Line execution environment results as the baseline for comparison. This resulted in the outcomes depicted in Table 4.1.

Table 4.1: False positives and false negatives in criteria applicability on *Command Line* execution environment

Criterion	False positives	False negatives
1.2.3		11%
1.2.8	2%	12%
1.3.1		27%
3.1.1		6%
3.2.2		9%
3.2.5	1%	5%
3.3.2		9%
3.3.5		6%
4.1.1		1%
4.1.2		37%

This analysis shows that, in fact, nearly 67% of the cases (10 criteria out of the 15 that were implemented) in the Command Line execution environment yield false negatives, i.e., were unable to be applied. The occurrence of false positives, i.e., when a Web page version for the Command Line execution environment triggered the application of criteria but not on the Browser execution environment, was substantially lower.

In the following sections will be detailed the four WCAG 2.0 criteria that reflect the different evaluation natures that emerge from the comparison of the outcomes of the two execution environment: *1.1.1*, *1.2.3*, *2.4.4*, and *3.2.2*.

WCAG 2.0 Criterion 1.1.1 Criterion 1.1.1 is the *poster child* of Web accessibility adequacy (both in engineering and evaluation terms). It reflects the necessity for content equivalence, thus enabling content understanding, no matter what impairment a user has. For instance, the existence of alternative textual descriptions for images. Thus, it was analysed individually this criterion, as depicted in Figure 4.10.

For a significant number of the Web pages analysed, there is a high increase of situations that could be detected in the Browser context. A brief glance at these differences showed the dynamic injection of images at either the *DOM Ready* or *DOM Load* browser

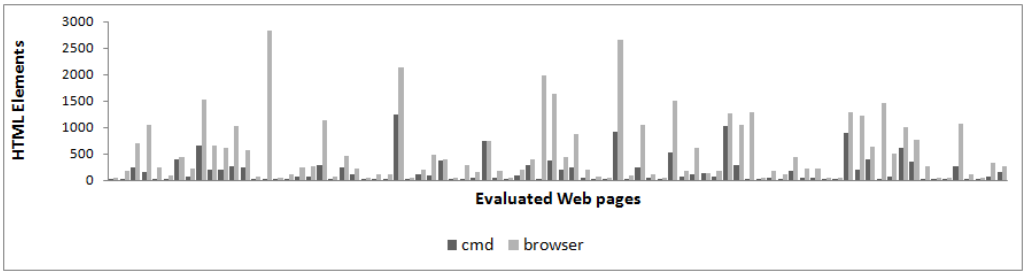

Figure 4.10: Browser vs Command Line for criterion 1.1.1

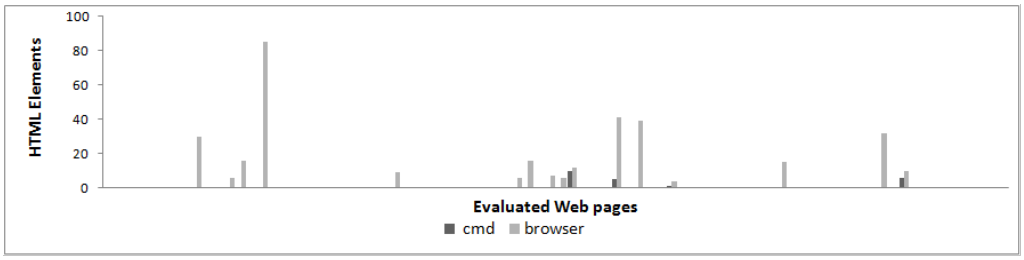

Figure 4.11: Browser vs Command Line for criterion 1.2.3

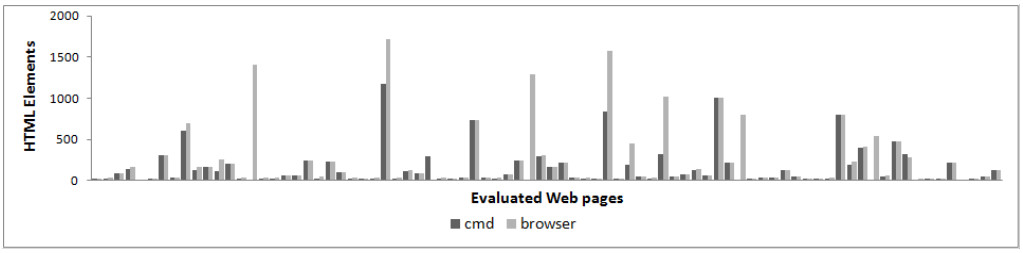

Figure 4.12: Browser vs Command Line for criterion 2.4.4

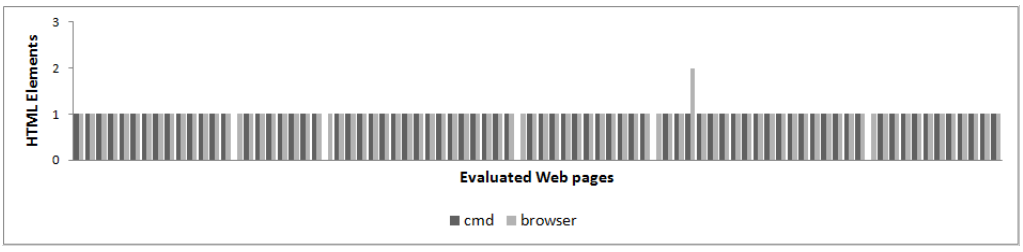

Figure 4.13: Browser vs Command Line for criterion 3.2.2

rendering events. This kind of disparity of the results is the one that occurs more often for all of the implemented criteria.

WCAG 2.0 Criterion 1.2.3 Criterion 1.2.3 depicts, in Figure 4.11, one case of the aforementioned false negatives. Almost all of the detected applicability occurred in the Browser execution environment.

WCAG 2.0 Criterion 2.4.4 In the case of Criterion 2.4.4, as depicted in Figure 4.12, most of the results are typical, i.e., at least an equal or greater number of elements are detected in the *Browser*. However, as identified in the graph, there is a Web page where the *Command Line* execution environment detects a substantially bigger amount of problems for this criterion. While not all of those cases disappear in the *Browser* execution environment, it shows that even when no false positive is raised for a criterion's applicability, there are cases where dynamic scripts remove detectable accessibility issues.

WCAG 2.0 Criterion 3.2.2 Finally, Criterion 3.2.2, as depicted in Figure 4.13, allows the detection of the (un)availability of form submission buttons. This could not be detected in the Command Line execution environment (i.e., the missing gaps in the graph), as these buttons were dynamically injected into the Web page.

4.2.4 Discussion

The study on the resulting outcomes from evaluating Web accessibility in the *Command Line* and *Browser* execution environments has yielded an interesting amount of insights, respecting to automated Web accessibility evaluation practices. It can be concluded that, due to the presented results the hypothesis, *H1*, was proven.

In the next sections, it will be discussed the Web accessibility evaluation in the Browser, finishing with a discussion of the limitations of the experimental setup.

Web Accessibility Evaluation in the Browser

The expectations with regards to the raised hypothesis (H1) were confirmed. In fact, there are deep differences in the accessibility evaluation between the *Command Line* and *Browser* execution environments. This is reflected not only in the additional amount of processable HTML elements, but also on the rate of the false negatives and positives yielded by Command Line execution environment evaluations as well.

Hence, it is important to emphasize that evaluating the accessibility of modern Web pages in a *Command Line* execution environment can deliver misleading paths for designers and developers, due to the following reasons:

- there are significant differences between the structure and content of Web pages in both execution environments. Thus, for dynamic Web pages, developers and designers can be faced with evaluation results that reflect different HTML DOM trees. This fact, on its own, can often provide confusion and result on difficulties of detecting the actual points where accessibility problems are encountered;

- False positives at the *Command Line* execution environment provide another point that can confuse designers and developers that are faced with these accessibility evaluation results, since they become invalid in the *Browser* execution environment (e.g., corrected with the aid of Javascript libraries);

- False negatives are most critical, since a lot of potential accessibility problems are simply not detected in the *Command Line* execution environment. Consequently, an evaluation result might pass on 100% of accessibility checks, but the HTML DOM tree that is presented to end-users faces severe accessibility problems.

These results show that, it is of the most importance to evaluate the accessibility of Web pages in the execution environment where end-users interact with them. The often proposed methodology of building Web pages in a *progressive enhancement* fashion (where scripts insert additional content and interactivity) do guarantee neither the improvement, nor the maintenance of the accessibility quality of any given Web page.

4.2.5 Limitations

The experiment has faced some limitations, both in terms of its setup, as well as on the type of results that can be extrapolated, including:

- *Data gathering*: since there were gathered all of the Web pages in the two execution environments at different instants, it could not be 100% guaranteed that the Web page generation artefacts were not introduced between requests for each evaluated Web page. Furthermore, the presented results are valid for the sample set of Web pages that were selected. However, it was believed that these pages are representative of modern Web design and development of front-ends;

- *DOM trees*: while the *QualWeb evaluator* takes a DOM representation of the HTML, it was only analysed the profusion of Web accessibility inadequacies in term of HTML elements, leaving out other potential factors that influence the accessibility of Web pages (e.g., CSS);

- *Comparison of DOM trees*: experimental setup did not provide enough information to pinpoint what transformations to the HTML DOM were made at both *DOM Ready* and *DOM Load* phases;

- *Script injection*: encountered some cases (for example: `facebook.com`) where the injection of accessibility evaluation scripts was blocked with *cross-site scripting* (XSS) dismissal techniques. In these cases, minimal alterations were hand-crafted on these Web pages, in order to disable these protections. For example: removal of some scripts that prevent code injection. Nevertheless, none of these alterations influenced the outcome of the accessibility evaluations performed in these cases;

- *Automated evaluation*: since this experiment is centred on automated evaluation of Web accessibility quality, it shares all of the inherent pitfalls. This includes the limited implementation coverage of WCAG 2.0.

4.3 Experimental Study 2 - Templates on Web Accessibility Evaluation

This experimental study was centered on analysing similarities of HTML elements between Web pages. The similarity criteria targets typical template-based definitions. The study was performed in a set of sites that feature a consistent use of HTML.

In the next section will be detailed the setup of this experiment, denote data acquisition and processing and present the most significant results from the experiment.

4.3.1 Setup

The selection rationale was to select well-known and representative Web sites from the Alexa's Top 100 Web sites[2] – *Google*, *Wikipedia*, *Facebook* and *Amazon* –, two modern online Portuguese newspapers – *DN* and *Público* – and the *WordTaps*. *WordTaps*'s Web site was chosen because it uses *WordPress*[3], a well known blogging and Web site platform. Table 4.2 presents the Web sites chosen.

Table 4.2: Analysed Web sites

`http://www.google.com`
`http://www.publico.pt`
`http://www.dn.pt`
`http://wikipedia.org`
`http://www.facebook.com`
`http://www.amazon.com`
`http://wordtaps.com`

[2]Alexa Top 100: `http://http://www.alexa.com/topsites`
[3]WordPress: `http://wordpress.org/`

4.3.2 Data Acquisition and Processing

It was selected a Web page from each Web site, other than the home page. Theses pages were then compared with the respective home page, to obtain the set of elements that are common between them (the *template* set) and the set that is specific for the Web page (the *specific* set).

Each Web page is then assessed using the automatic *QualWeb evaluator* (developed in the beginning of this work) and the reported errors are matched with the elements in the above-mentioned set. This division allows a faster access to each type of accessibility evaluation results. The process was repeated for all the Web sites.

4.3.3 Results

The study was focused on the percentage of WCAG 2.0 techniques applicability (i.e., specific outcomes - pass, warn, fail). The average of all the template sets is 38.85% ($\sigma = 7.48$), and in the specific content is 61.15% ($\sigma = 7.48$). Besides, the averages for the outcomes considered in the applicability are: 34.50% of warnings ($\sigma = 7.00$), 0.80% of fails ($\sigma = 1.00$), and 3.56% of pass ($\sigma = 2.64$). The percentage of errors that need to be repaired in the templates have an average of 38.06% ($\sigma = 7.78$).

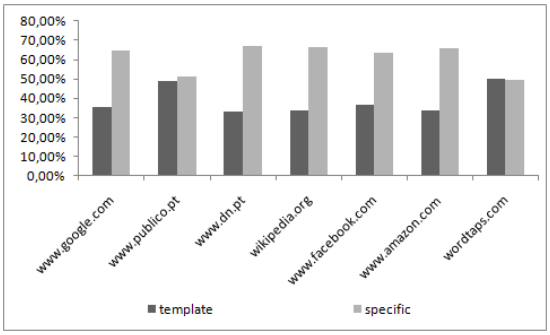

Figure 4.14: Applicability of WCAG 2.0 techniques on one of the evaluated Web pages.

4.3.4 A Template-Aware Web Accessibility Metric

Figure 4.15 contains 7 examples of applicability for Web sites sample, considering the Web page template and the specific part of the Web pages considered. The charts of the figure show variable applicability in the specific parts of Web pages, but higher than template applicability, as expected because of the results obtained previously. These results are similar for all examples.

In the next section will be presented the results of the metric application in two Web sites.

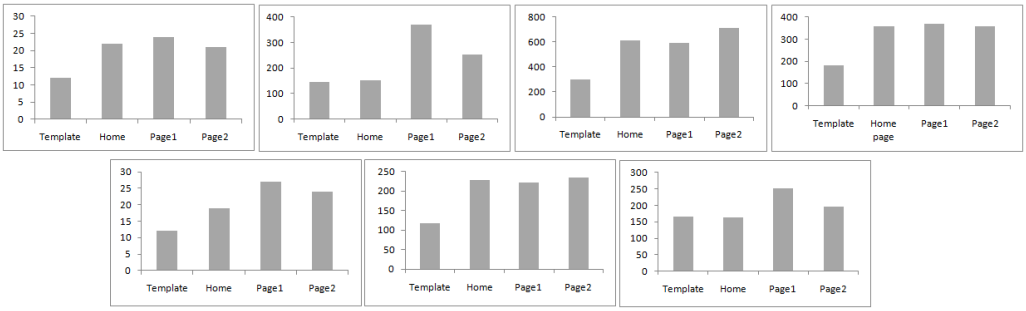

Figure 4.15: Graphs represent the elements per Web page, on top row left to right the Web sites are: *Google*, *Público*, *DN* and *Wikipedia*. In bottom row the Web site are: *Facebook*, *Amazon* and *Wordtaps*

Results of Metric Application

Two simple examples of the application of the metric were performed with *Google* and *DN*. For these examples, the Web sites were evaluated considering the home page and the Web pages of the same domain directly accessed by it. Besides to simplify the test, the sub-menus were not considered.

Equation (4.1) shows an example of the application of the metric in the *Google* Web site.

$$\alpha(Google) = \begin{cases} 12 \\ [22, 24, 32, 1204, 1405, 179, 987, 22, 22, 97, 295, 113, 178, 75, 17, 72] \end{cases} \tag{4.1}$$

Equation (4.2) shows an example of the application of the metric in the *DN*.

$$\alpha(DN) = \begin{cases} 1741 \\ [3060, 3415, 3275, 3510, 3873, 3761, 3445, 3896, 3049, 3744, 3951, \\ 3626, 3363, 3992, 2166, 3311, 3736, 3945, 3633, 3341, 3431, 4021, \\ 3652, 3846, 3523, 3261, 3705, 3786, 3976, 3709, 3699, 3333, 4044, \\ 3607, 3739, 4031, 3785, 3843, 3702, 3806, 3924, 3716, 3599, 3778, \\ 3665, 3653, 3590, 3967, 3712, 3970, 3123, 3775, 3899, 3966, 3548, \\ 3671, 3566, 3561] \end{cases} \tag{4.2}$$

It can be verified that in the first example, there is a small number of elements in the template applicability. This happens because, in the *Google* Web site, some pages are very similar (e.g., home page, image search engine and video search engine), but there are other pages that maintain few similarities. In a newspaper Web site, as DN, there are a larger number of similarities between the Web pages.

4.3.5 Discussion

The presented results point to a positive verification of the second hypothesis, *H2*. In the next sections, it will be discussed the impact of the Web page templates in several thematic, as well as the experiment limitations.

Impact on Evaluation Results

It was determined that approximately 39% of the results are reported at least twice, of which approximately 38% are errors that can be corrected once.

Taking into account that templates are often automatically generated, template accessibility errors are automatic produced. Errors in the specific content of a Web page depend on the accessibility knowledge of the designer/developer who is developing the Web site and on how good some helping tools, that they can use, are. However, as it can be seen in the results, this type of errors has the highest percentage.

This could be developed even further considering the similarities between elements on more than two pages, and determining the number of times (more than two) that a WCAG 2.0 technique is applicable to each common element on the site. As such, reporting can be additionally simplified, performance can be improved, and more accurate metrics can be defined.

Besides, regarding repairing, template aware reports can be integrated in development tools directing developers/designers to a much more effective error correction process.

Finally, it is important to understand the differences in the Web pages of a Web site. Because, as happens in *Google*, differences can be very paradigmatic, i.e., the Web page templates can be very diverse in the same Web site. This way, when the template detection is performed, it is important to get maximum coverage of pages, i.e., combine the pages in various ways to get the templates most suitable among them.

Impact on Web Accessibility Research

The results obtained shown that it is possible for accessibility experts and researchers to create new accessibility metrics (as the metric created in this work), that can, for example, measure the template influence on Web pages and/or Web sites. Besides, they can create new and different evaluators that simplify accessibility evaluations using templates.

Assessment methods should be modified in order to do not consider Web pages as a whole. This way, pages can be divided, as suggested, in template and specific sets to improve evaluations and consider their various characteristics.

Impact on Designing Accessible Web Pages

Designers can access to accessibility evaluation results report, without the same template errors repeated for each Web site page. Therefore, they only have to repair a problem

once and, because of the template use, this problem should remain corrected throughout the Web site. Consequently, repair of accessibility errors can be simplified and time spent in design can be reduced.

In some cases, developers might encode similar/equal elements in multiple Web pages – *extra templates* — which makes them being identified as part of the Web page template. However, its correction is not propagated automatically. If developers realize that these templates are often used, they can be included in the Web page template. This inclusion will facilitate the repair of its errors and help the design of accessible Web pages.

4.3.6 Limitations

The experiment has faced some limitations, in terms of template detection and the selected Web sites for the experimental study:

- *Intra-page template*: intra-page templates are defined inside the Web page itself such as list, ads, etc. Since this type of templates are not considered, all the possible repetitions of accessibility errors for these cases were not removed.

- *Thresholds*: more tests were performed to define the threshold values, nonetheless these values may be incorrect for some Web sites not tested. Because of that, they may exclude some elements that are part of the template or the opposite;

- *Web sites sample*: the study focuses only on the seven Web sites selected. However, these Web sites are representative of the best practices of template usage. Additionally, although non-template Web sites were not the focus, the expected result is that the *Fast Match* algorithm could not find any match or it finds very few. This assumption can be made, because in the test phase of the algorithm two completely different trees were tested and it did not present any match;

4.4 Summary

This chapter described the experiments carried out in this work. The accessibility evaluation framework performed was used to evaluate the accessibility of Web pages of some of the most popular Top 100 Web sites, in two different execution environments (Command Line and Browser). The results of these evaluations show the advantage of the evaluation to be held in the transformed HTML, proving the hypothesis. Besides, there is a big percentage of false negatives and a lower percentage of false negatives.

Relatively to the use of Web page template in accessibility, a template detection algorithm was used to verify if the repair of accessibility errors was facilitated. It was shown that the accessibility results of the common elements are more than a third of the whole results set. A significant percentage of the accessibility errors that would simplify error

reports and consequently the developers/designers work. This way, developers/designers can repair accessibility errors only once and these are automatically repaired throughout the Web site.

Chapter 5

Conclusion

All the objectives proposed for this work were accomplished and all the identified requirements for accessibility evaluation were considered. An automatic Web accessibility framework, able to perform evaluation in both envisaged execution environments, was designed and developed. The WAE framework uses the more recent accessibility standard guidelines, for which a significant subset of techniques was implemented. A test-bed was also produced to validate the implemented techniques. Moreover, a reporting mechanism using EARL was implemented, ensuring compatibility with official reporting standards. The whole framework architecture ensures easy extensibility for: a) emerging reporting standards and specific ones that assure integration with complementary development tools; b) new techniques derived from different and alternative accessibility guidelines. Finally, the whole WAE was designed considering the common Web development procedures. For that, the framework incorporates the generation of accessibility assessment results and a new metric, both template-aware.

Relatively to the first experimental study presented, it was performed an automated WAE in the context of two execution environments: *Command Line* and *Browser*. For the experiment, it was analysed the accessibility quality of the home pages of the Top 100 most visited Web sites in the world. It was provided evidence that the significant differences introduced by AJAX and other dynamic scripting features of modern Web pages do influence the outcome of Web accessibility evaluation practices. It was showed that the automated WAE in the *Command Line* execution environment can yield incorrect results, especially on the applicability of success criteria.

Further, it was performed and presented a second experimental study on detection of Web page templates to facilitate the repair of accessibility errors. For the experiment, a subset of Web pages of 7 different Web sites was analysed. It shown that the accessibility results of the common elements are more than a third of the whole results set. This is a significant percentage of the accessibility errors, which would help to improve reporting and consequently simplify the developers/designers work.

Both studies confirm the initial hypothesis that: 1) WAE should be performed after the

browser processing, as a means for more accurate evaluation; and 2) that it is possible to simplify reporting and devise metrics that, in conjunction, provide better departing points for web site repairing.

5.1 Future Work

Facing with the obtained results of the comparative of *Command Line* and *Browser* environments, and based on the implementation of the *QualWeb evaluator* and environment evaluation framework, some work can be conducted in the following directions:

1. implementation of more WCAG 2.0 tests based on the analysis of CSS, especially in the post-cascading phase, when all styling properties have been computed by the Web browser;

2. continuous monitoring of changes in the HTML DOM, thus opening the way for detection of more complex accessibility issues, such as WAI ARIA live regions [20];

3. detecting the differences in DOM manipulation, in order to understand the typical actions performed by scripting in the browser context;

4. the implementation of additional evaluation environments, such as developer extensions for Web browsers (e.g., Firebug[1]), as well as supporting an interactive analysis of evaluation results embedded on the Web pages themselves.

Besides, looking at the obtained results in template detection, some work can also be conducted in the following points:

1. explore intra-page templates, that will allow to detected a greater percentage of templates and reduce the inner page repetitions of the same accessibility errors;

2. explore extra templates, it can be suggested to developers the inclusion of this extra templates in the Web page templates;

3. a larger sample of Web sites should be used, for template detection;

4. compare more than two pages and therefore errors that will be reported more than two times;

5. assess the Fast Match algorithm in order to fully understand how accurately it matches the template, i.e., what elements are actually components of a template and which are not.

Finally, it is planned to provide all the code online in the GitHub repository[2].

[1]Firebug: `http://getfirebug.com/`
[2]GitHub repository: `https://github.com/`

Appendix A

Papers Written

A.1 Avaliação Pericial de Barreiras ao Acesso sobre Sítios Web de Entidades Públicas - Interacção 2010

Avaliação Pericial de Barreiras ao Acesso sobre Sítios Web de Entidades Públicas

Nádia Fernandes
Universidade de Lisboa
nadia.fernandes@di.fc.ul.pt

Rui Lopes
Universidade de Lisboa
rlopes@di.fc.ul.pt

Luís Carriço
Universidade de Lisboa
lmc@di.fc.ul.pt

Sumário

A acessibilidade de sítios Web é um factor crucial para pessoas com deficiências, para que estes consigam aceder a informação relevante existente em páginas Web. Este artigo apresenta uma análise pericial efectuada ao sítio Web do Governo de Portugal (http://www.portugal.gov.pt) baseada na metodologia de detecção de barreiras ao acesso Barrier Walkthrough. Mostramos que este tipo de metodologias potencia a detecção de um número maior de problemas, comparativamente aos processos normalmente utilizados.

Palavras-chave

Acessibilidade Web, Avaliação Pericial, Barrier Walkthrough.

1. INTRODUÇÃO

O conceito de acessibilidade baseia-se na facilidade de acesso a conteúdos ou serviços por parte de pessoas com algum tipo de incapacidade sem que para isso necessitem do auxílio de terceiros.

A directiva em Portugal que obriga à acessibilidade de sítios Web institucionais é a Resolução do Conselho de Ministros número 155/2007 [PCM07], que utiliza a norma WCAG 1.0 [Chisholm99]. De acordo com esta directiva, "a organização e apresentação da informação facultada na Internet pelos sites do sector público", devem ser escolhidas para permitir ou facilitar o seu acesso pelos cidadãos com necessidades especiais. A acessibilidade deverá abranger, no mínimo, a informação relevante para a compreensão dos conteúdos e para a sua pesquisa".

Apresentamos um estudo da acessibilidade no sítio do Governo de Portugal, para utilizadores: invisuais, daltónicos e com deficiências nos membros superiores.

2. TRABALHO RELACIONADO

2.1 Avaliação Pericial da Acessibilidade Web

A avaliação pericial da acessibilidade da Web pode ser realizada tendo em conta que é definido em normas como a WCAG, ou seguindo metodologias de análise pericial como a Barrier Walkthrough [Brajnik09].

A Barrier Walkthrough é uma metodologia em que estão definidas categorias de utilizadores e as barreiras para cada categoria. Sendo as barreiras qualquer condição que impeça um utilizador com uma deficiência de cumprir um objectivo. Assim, para cada categoria de utilizadores são verificadas as barreiras que se verificam e que o impeçam o seu acesso ao sítio. Podendo verificar-se se os critérios da WCAG 2.0 estão a ser cumpridos.

2.2 Avaliações Institucionais

A Resolução do Conselho de Ministros número 155/2007 indica que se deve assegurar que a informação disponibilizada pela Administração Pública na Internet seja acessível a cidadãos com necessidades especiais. Na actualidade, o acesso às tecnologias da informação e da comunicação e a capacidade para a sua utilização são diferenciadores das oportunidades sociais.

O logótipo de "Certified Accessibility" da UMIC [ASC09] indica a acessibilidade do sítio em que se encontra**.** É dinâmico e permite a vigilância dos conteúdos de um sítio na Web, através do validador eXaminator (utiliza WCAG 1.0). Apresenta vários estados de acordo com o cumprimento das normas. Esta avaliação é automática, não detectando muitos dos erros detectados com o Barrier Walkthrough, nem os agrupando por deficiências.

A existência de mais de 600 mil pessoas com incapacidades em Portugal (EU 2002) que são "info-excluídas", suportam a ideia de que criar sítios Web acessíveis.

Em 2008, foram realizados estudos de acessibilidade dos sítios Web das mil maiores empresas portuguesas em volume de negócio [Gonçalves09] (INE 2007), seguindo-se as normas do W3C, pelo Grupo de Negócio Electrónico. Os resultados foram: 9,4% apresentam o nível A, uma tem o nível de AA e nenhuma apresenta o nível máximo.

3. AVALIAÇÃO PERICIAL

3.1 Metodologia

Seguiu-se a metodologia Barrier Walkthrough, para as categorias de utilizadores/deficiências já referidas. Escolheram-se vinte e cinco templates representativos dos conteúdos e estruturas do sítio do governo. O critério de escolha dos templates escolhidos foi a diferença da estrutura entre eles e a verificação visual de barreiras.

4. RESULTADOS

A avaliação resultou na detecção de diversos problemas de acessibilidade. Obteve-se uma taxa de aprovação de 30% para invisuais, de 50% para daltónicos e de 50% para deficiências dos membros superiores.

De seguida vão ser descritos algumas das barreiras encontradas e possíveis formas de eliminação das mesmas.

3.2.1 Links genéricos

São links que não fornecem informação suficiente para que se compreenda o seu conteúdo. Por exemplo, um link com o texto "mais… " que nos redirecciona para uma página de notícias (Figura 1). Seleccionar-se-ia este problema modificando os labels de links para que dessem pistas da página que vai ser aberta.

```
<a
  href="/pt/GC18/Noticias/Pages/20100608_Not_CM
  _PMEInveste.aspx"> mais...
</a>
```

Figura 1 – Exemplo no código de um link genérico

3.2.2 Objectos opacos

São componentes totalmente opacos para os leitores de ecrã. Nesta caso, são utilizados vídeos em Flash (Figura 2). Isto poderia ser resolvido garantido que o objecto é acessível, seguindo directivas específicas, caso isso não fosse possível o objecto deveria ser removido.

```
swfobject2.embedSWF(
"/pt/GC18/ConteudosTransversais/Flashs/20091118_
Governo/slideshowpro.swf",
 "flashcontent","978","255",   "9.0.0",   false,
flashvars, params, attributes);
```

Figura 2 – Exemplo no código de um objecto opaco

3.2.3 Eventos do rato

São eventos desencadeados apenas com a utilização do rato, pessoas que não utilizassem o rato nunca conseguiriam utilizá-los. Isto verifica-se na utilização de event handlers ("onclick", ...) que são orientados para o rato. O problema era resolvido utilizando event handlers lógicos ("onfocus", ...) além dos orientados para o rato.

3.2.4 Nova janela

Quando se clica num link somos redireccionados para uma nova janela no browser, devido à utilização de ´target="_blank"´. O problema seria resolvido evitando abrir-se novas janelas normalmente. Se for mesmo necessário deve haver um link ou botão que permita fechar a janela, para que os utilizadores percebam que se abriu uma nova janela e que têm a possibilidade de a fechar.

3.2.5 Não é possível saltar links

Não é permitido saltar directamente para o conteúdo da página. Por exemplo o utilizador tem de passar por todos os links anteriores antes de chegar ao link que pretende.

3.2.6 Contraste visual insuficiente

A página contém elementos cujo contraste entre estes e o fundo é insuficiente. Por exemplo, o contraste entre fundo branco e ícones/texto cinzento. Este problema poderia ser resolvido com o aumento do contraste dos elementos.

5. DISCUSSÃO

Apesar dos esforços para tornar o sítio do governo mais acessível, encontraram-se várias barreiras de acesso, que impedem os utilizadores de aceder correctamente aos conteúdos e de realizarem operações. Por exemplo, os invisuais poderiam ter de percorrer toda a página para encontrar um link e pessoas com deficiências nos membros superiores não evitariam a utilização do rato.

Sendo as metodologias empregues insuficientes, deveriam ser tomadas mais medidas para que o site fosse acessível a todos os seus utilizadores. A resolução conselho ministros, não está a ser cumprida e está desactualizada, utilizando ainda WCAG 1.0, quando a norma suportada pelo do W3C é a WCAG 2.0.

6. CONCLUSÕES E TRABALHO FUTURO

A acessibilidade na Web é importante para que as pessoas com deficiências consigam compreender conteúdos e realizar as actividades que pretendem. Com a análise pericial realizada ao sítio do Governo de Portugal, baseada na metodologia Barrier Walkthrough, verificou-se que este tem bastantes barreiras ao acesso.

No seguimento deste trabalho, aplicaremos este processo no estudo de outros sítios Web de instituições públicas. Procederemos à comparação quantitativa de processos de avaliação pericial com avaliadores automáticos e à exploração destas limitações no âmbito de testes de acessibilidade com utilizadores finais.

7. REFERÊNCIAS

[ASC09] Agência Para A Sociedade Do Conhecimento, Programa Acesso, 20 de Setembro de 2009, http://www.acesso.umic.pt/webax/nota_tecnica_logo.html

[Brajnik09] Brajnik, Giorgi, Barrier Walkthrough Heuristic Evaluation Guided by Accessibility Barriers, Março de 2009, http://users.dimi.uniud.it/~giorgio.brajnik/projects/bw/bw.html

[Chisholm99] Chisholm, Wendy, Vanderheiden, Madison Gregg, Jacobs, Ian, Web Content Accessibility Guidelines (WCAG) 2.0, 5 de Maio de 1999, http://www.w3.org/TR/WCAG10/

[Gonçalves09] Gonçalves, Ramiro, Pereira, Jorge, Martins, José, Mamede, Henrique, Santos, Vítor, Web Ponto de Situação das Maiores Empresas Portuguesas, Setembro 2009, http://www.acesso.umic.pt/estudos/1000maioresempresas_apdsi_0909.pdf

[PCM07] Presidência Do Conselho De Ministros, Resolução Do Conselho De Ministros N.º 155/2007, Diário Da República, 1.ª série — N.º 190 — 2 de Outubro de 2007, http://www.umic.pt/images/stories/publicacoes200710/RCM%20155%202007.pdf

A.2 On Web Accessibility Evaluation Environments - W4A 2011

On Web Accessibility Evaluation Environments

Nádia Fernandes, Rui Lopes, Luís Carriço
LaSIGE/University of Lisbon
Campo Grande, Edifício C6
1749-016 Lisboa, Portugal
{nadia.fernandes,rlopes,lmc}@di.fc.ul.pt

ABSTRACT

Modern Web sites leverage several techniques (e.g. DOM manipulation) that allow for the injection of new content into their Web pages (e.g., AJAX), as well as manipulation of the HTML DOM tree. This has the consequence that the Web pages that are presented to users (i.e., *browser* environment) are different from the original structure and content that is transmitted through HTTP communication (i.e., *command line* environment). This poses a series of challenges for Web accessibility evaluation, especially on automated evaluation software.

This paper details an experimental study designed to understand the differences posed by accessibility evaluation in the Web browser. For that, we implemented a Javascript-based evaluator, *QualWeb*, that can perform WCAG 2.0 based accessibility evaluations in both *browser* and *command line* environments. Our study shows that, in fact, there are deep differences between the HTML DOM tree in both environments, which has the consequence of having distinct evaluation results. Furthermore, we discovered that, for the WCAG 2.0 success criteria evaluation procedures we implemented, 67% of them yield false negative answers on their applicability within the *command line* environment, whereas more than 13% of them are false positives. We discuss the impact of these results in the light of the potential problems that these differences can pose to designers and developers that use accessibility evaluators that function on *command line* environments.

Categories and Subject Descriptors

H.5.4 [**Information Interfaces and Presentation**]: Hypertext/Hypermedia—*User issues*; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Evaluation/methodology*; K.4.2 [**Computers and Society**]: Social Issues—*Assistive technologies for persons with disabilities*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
W4A2011 - Technical Paper, March 28-29, 2011, Hyderabad, India. Co-Located with the 20th International World Wide Web Conference.
Copyright 2011 ACM 978-1-4503-0476-4 ...$5.00.

General Terms

Measurement, Human Factors.

Keywords

Web Science, Web Accessibility, Web Accessibility Evaluation Environments, Automated Evaluation.

1. INTRODUCTION

Accessibility on the Web is often framed in a tripartite way: Web page semantics, assistive technology (AT), and Web browser capabilities. Given an arbitrary Web page, its content is exposed by the Web browser in such a way that AT aids users with disabilities understanding and interacting with it. Existing best practices for Web accessibility adequacy are based on these two factors: WCAG [4] defines best practices for Web page semantics, whereas UAAG [6] dictates how Web browsers must be implemented in order to leverage AT.

These best practices can also be applied as checklists for evaluation. In the case of WCAG, they can be used to assess how accessible a Web page is. This evaluation procedure can be performed (1) with users, such as usability tests, (2) through expert analysis, and (3) with the aid of automated evaluation software. While usability tests and expert analysis are focused on the rendered state of the Web page within the browser, most implementations of automated evaluation just focus on the Web page content that is sent through the first HTTP request.

With the ever growing dynamics of Web pages (e.g., AJAX and other Javascript techniques), the state of a Web page's content, structure, and interaction capabilities are becoming different in what regards to their initial HTTP communication. Several dynamic content techniques allow for displaying/hiding information, injecting new content, and even removing content from Web pages. Since AT is capable of interacting with this kind of content through modern Web browsers, it is imperative for automated evaluation to be applied to the content Web browsers display.

Following this line of thought, this paper presents an experimental study on automated evaluation of Web accessibility at two different evaluation environments: *Command Line* – representing the typical environment for automated evaluation, which includes existing evaluators that can be accessed online – and *Browser*, the environment where users interact with the Web. Our study centres on the application of the same implementation techniques for evaluating a representative subset of WCAG 2.0, to understand

the impact of evaluating the accessibility of Web pages in the *browser* environment. Next, we discuss the typical Web browsing process that happens when end-users interact with Web pages.

2. WEB BROWSING PROCESS

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

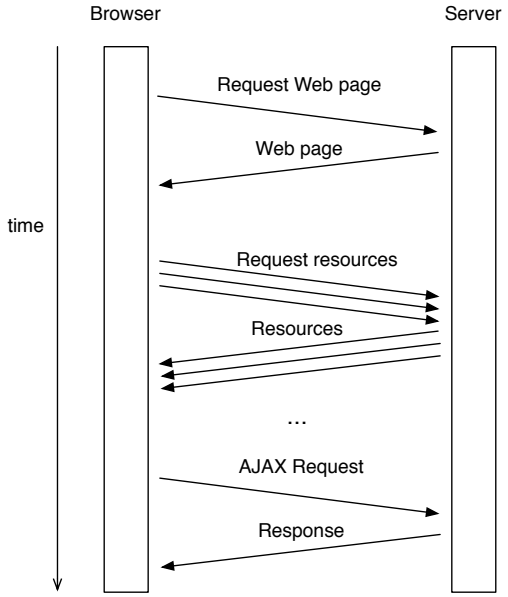

Figure 1: Web Browsing Resource Interaction

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

- *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.

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*). Next, we further detail these aspects.

2.1 Architecture of the Web

The architecture of the Web [9] is composed by servers, URIs, and user agents. User agents (such as Web browsers) communicate with servers to perform a retrieval action for the resource identified by the URI. A server responds with a message containing a resource representation. As depicted in Figure 1, in the case of Web browsers, a Web page is represented not just by its HTML content, but also by a set of ancillary resources. Due to this increased complexity on handling resources and their representation for users, Web browsers process all the resources through adequate technologies (e.g., executing Javascript), which results in the transformed HTML document that is presented to users.

2.2 Web Page Loading Process

After all resources are successfully delivered to the Web browser, four steps are sequentially executed before users are able to interact with the Web page, as depicted in Figure 2:

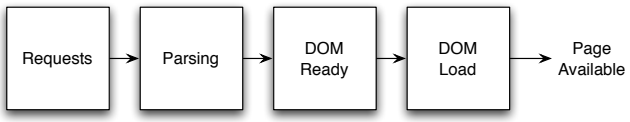

Figure 2: Web Page Loading Process

The first step in the Web page loading process, *Requests*, concerns getting all resources that compose the Web page. After that, the Web browser *parses* these resources, i.e., build the HTML DOM tree, the CSS object model, and constructing the execution plan based on the existing scripted behaviours. Afterwards, the browser triggers two events in sequence: *DOM Ready* and *DOM Load*. The former is triggered when the HTML DOM tree is ready, whereas the second is triggered after all resources are ready (CSS, images, etc.)

Web pages typically 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 potentiate the addition/removal/transformation of this tree. Consequently, the Web page a user is presented might be from slightly too heavily different from the URI's resource representation that is initially transmitted to the browser from the Web server.

2.3 Research Hypothesis

In the light of the way browsers interpret Web pages, as detailed above, and taking into account that users with disabilities interact with these Web resources through browsers and AT, we devised the following research hypothesis that serves as the basis for our experimental study:

> *Evaluating Web content in the browser provides more accurate and more in-depth analysis of its accessibility.*

To investigate the outcome of this hypothesis, we established the following assumptions: (1) there is the need for understanding what are the differences in the HTML between environments; (2) discover the limitations of accessibility evaluation in different environments; (3) evaluation procedures must be the same in all environments for we can compare them.

Next, in the light of this hypothesis and corresponding assumptions, we present the related work on Web accessibility evaluation particularly focusing on automated evaluation procedures, as well as in-browser evaluations.

3. RELATED WORK

To help create accessible Web pages, WCAG define guidelines that encourage 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 [8, 11]. Although these guidelines exist and are supposed to be followed by the creators, most Web sites still have accessibility barriers making its utilization very difficult or even impossible for many users [8]. Thus, WCAG can also be used as a benchmark for analysing the accessibility quality of a given Web page.

Web Accessibility Evaluation is an assessment procedure to analyse how well the Web can be used by people with different levels of disabilities [8]. Optimal results are achieved with combinations of the different approaches of Web accessibility evaluation, taking advantage of the specific benefits of each of them [8]. Therefore, conformance checking [2], e.g., with the aid of automated Web accessibility evaluation tools can be an important step for the accessibility evaluation.

3.1 Automated Accessibility Evaluation

Automated evaluation is performed by software, i.e., it is carried out without the need of human intervention, which has the benefit of objectivity [11]. However, this type of assessment has some limitations as described in [10]. To verify where and why a Web page is not accessible it is important to analyse the different resources that compose the Web page. This analysis brings the possibility of measuring the level of accessibility of a Web page, with the aid of automated Web accessibility evaluation software. Examples include Failure Rate [12], UWEM [13], and WAQM [14].

3.2 Accessibility Evaluation in the Browser

In the past, the predominant technologies in the Web were HTML and CSS, which resulted in *static* Web pages. Today, on top of these technologies, newer technologies appear (e.g., Javascript), and, consequently, the Web is becoming more and more dynamic. Nowadays, user actions and/or automatically triggered events can alter a Web page's content. Because of that, the presented content can be different from the initially received by the Web browser.

However, automatic evaluations do not consider these changes in the HTML document and because of that results could be wrong and/or incomplete. Since expert and user evaluation are performed in the browser, they do not suffer with these changes. To solve this problem, the accessibility evaluation should be applied to new environments, i.e., in the Web browser context.

The importance of the Web browser context in the evaluation results is starting to be considered and is already used in three tools named *Foxability*, *Mozilla/Firefox Accessibility Extension*, *WAVE Firefox toolbar* [7] and the list of tools provided by Web Accessibility Initiative (WAI) [1]. However, these tools focus only evaluating Web pages according to WCAG 1.0. Furthermore, since the fist three evaluation procedures are embedded as extensions, they become more limited in terms of their application in the *command line* environment.

Also, since these tools focus on providing developer-aid on fixing accessibility problems, the resulting outcomes from evaluations are user-friendly, thus less machine-friendly. Therefore, if taking into account the proposed goal of this paper, it becomes cumbersome to define an experiment that can leverage the evaluation knowledge embedded in these tools. This *browser paradigm* – as called in [7] – is still nascent. Until now, to the best of our knowledge, differences between results in different evaluation environments are not clear. To perform correct comparisons, it must be guaranteed that tests are implemented in different environments in the same way, by reducing implementation bias.

Furthermore, we wanted to make a fair comparison between HTML pre and pos-processors evaluators. Having a single framework, provided that capability.

4. WEB ACCESSIBILITY EVALUATION ENVIRONMENTS

Our study is emphasized in two main environments: *Command Line*, and *Browser*. In the *Command Line* environment, evaluation is performed on the HTML document that is transmitted initially in an HTTP response, whereas in the *Browser* environment, evaluation is targeted at the transformed version of the HTML document.

Consequently, to better grasp the differences between these environments, we defined an architecture that allows for leveraging the same evaluation procedures in any environment, as detailed below. Afterwards, we explain how we implemented the ideas from this architecture, as well as how it was validated.

4.1 Architecture

The architecture of the evaluation framework is composed by five components, as depicted in Figure 3: the QualWeb Evaluator, Environments, Techniques, Formatters, and Web Server.

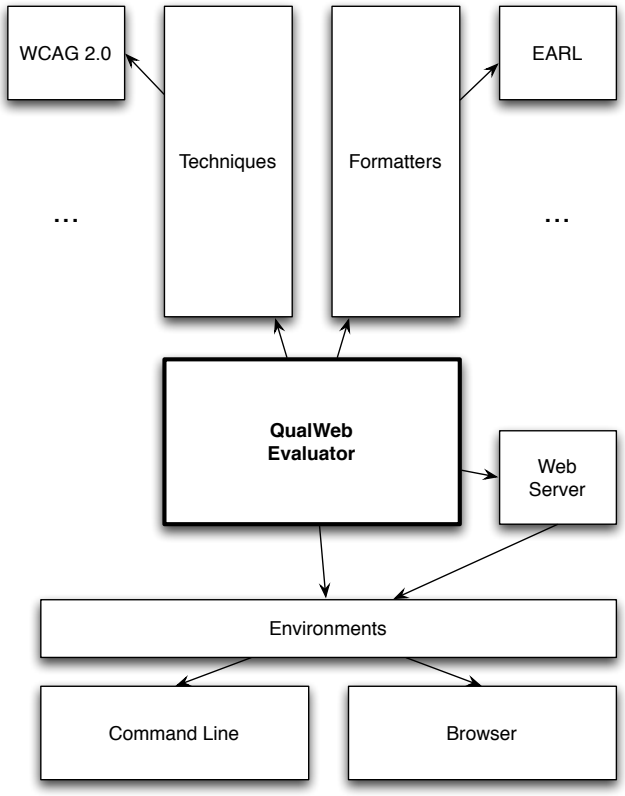

Figure 3: Architecture of the Evaluation Framework

The *QualWeb Evaluator* is responsible for performing the accessibility evaluation of Web pages, through the features provided by the *Techniques* component (e.g., implementation of WCAG 2.0 techniques); it uses the *Formatters* component to tailor the results into specific serialisation formats, such as EARL reporting [3]. Finally, the QualWeb Evaluator is applied in the different *Environments*.

Finally, the *Environments* component instantiates the types of environments that can leverage the QualWeb evaluator. In the case of the *Browser* environment, we specified the requirement for a *Web Server* component, to allow for transmitting all evaluation assets (e.g., scripts) that are to be applied in the currently selected Web page, as well as to gather the evaluation results at a well-known point within the server.

4.2 Implementation

To facilitate the accurate replication of the experiment and to provide in-depth guidance on how to implement such evaluators we provide a high detail of the implementation.

In order to compare the proposed evaluation environments, we must use the same accessibility evaluation implementation. Given that one of the environments is the Web browser, we have a restriction on using Javascript as the implementation language. Thus, to develop the *Command Line* version of the evaluation process, we leveraged *Node.js*[1], an event I/O framework based on the V8 Javascript engine[2]. In addition to standard *Node.js* modules, we used several other ancillary modules[3], including:

- *Node-Static*, which allowed for serving static files into the browser environment;
- *Node-Router*, a module that supports the development of dynamic behaviours, which we used to implement the retrieval and processing of evaluation results, and
- *HTML-Parser*, which provides support for building HTML DOM trees in any environment.

Besides these standard modules, we also implemented a set of modules for our evaluation framework, including:

- *EARL module*, which allows for the creation of EARL documents with the defined templates and parse EARL files using the *Libxmljs* library, and
- *Evaluator module*, which performs the accessibility evaluation with the implemented techniques.

Next, we present additional details on how we implemented both evaluation environments, as well as report generation and processing capabilities.

4.2.1 Command Line Environment

This environment obtains the HTML document from a URL using an HTTP request, executes the *QualWeb evaluator* on the HTML DOM tree, and serialises its outcome into EARL. All of these processes are implemented with a combination of the *HTML-Parser*, *EARL*, and *Evaluator* modules, executed from a command line.

[1]Node.js: `http://nodejs.org`
[2]V8 Javascript engine: `http://code.google.com/p/v8/`
[3]GitHub modules: `https://github.com/ry/node/wiki/modules`

4.2.2 Browser Environment

This environment uses a *bookmarklet* (Figure 4) to trigger the execution of the evaluation within the browser. Bookmarklets are browser bookmarks that start with the `Javascript:` protocol. In front of this, pure Javascript commands follow. When a user activates the bookmarklet, these commands are run.

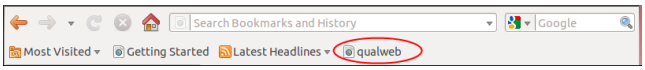

Figure 4: Evaluation execution example on Browser

In the case of our evaluator, this bookmarklet injects the necessary functions to obtain the HTML DOM tree of the current Web page, executes the QualWeb evaluator, and sends the evaluation results to a server component. These results are transformed in the EARL serialisation format, and subsequently stored. To implement this *browser-server* execution and communication mechanism, we used the following modules:

- *Bootstrap*, to import the required base modules, and
- *LAB.js*, to inject all of the evaluation modules into the browser's DOM context.

4.2.3 Report Generation and Processing

Finally, to generate the evaluation reports containing the accessibility quality results, we used the following modules:

- *Node-Template*, to define EARL reporting templates,
- *Libxmljs*, to parse EARL reports, and
- *CSV module*, to recreate a *comma-separated-values* (CSV) counterpart from a given EARL report. This module allowed for a better inspection and statistical analysis with off-the-shelf spreadsheet software.

While the EARL format allows for the specification of evaluation results, we had to extend EARL with a small set of elements that could allow for the analysis of the resulting outcomes from our experiment. Hence, we defined a *Metadata* field that supports the specification of *HTML element count*, as well as a *Timestamp* to state the specific time when the evaluation was performed.

The EARL reports served as the basis for generating CSV reports. Due to the extensiveness of EARL reports generated by our evaluator, specially in what respects to parsing and consequent memory consumption provided by generic DOM parsers, we implemented the EARL-CSV transformation procedures with SAX events.

4.3 Testability and Validation

We developed a test bed comprising a total of 102 HTML documents, in order to verify that all the WCAG 2.0 implemented techniques provide the expected results. They were based on documented WCAG 2.0 techniques and ancillary WCAG 2.0 documents. Besides, each HTML document was carefully hand crafted and peer-reviewed within our research team, in order to guarantee a high level of confidence on the truthfulness of our implementation. Success or failure cases were performed for each technique, to test all the possible

techniques outcomes. To get a better perspective on the implementation of our tests, we leveraged the examples of success or failure cases described for each WCAG 2.0 technique.

The graph depicted in Figure 5 shows the number of HTML test documents defined for each technique that was implemented in the QualWeb evaluator.

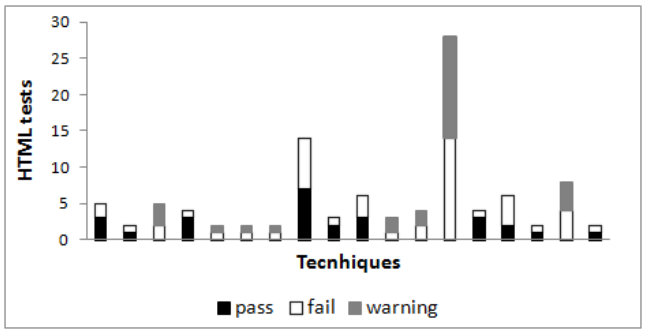

Figure 5: Number of Test Documents per Technique

We opted for having the same HTML documents, so that we could ensure that the evaluation outcomes aren't modified when changing evaluation environments. To test the proper application of the implemented techniques in the two evaluation environments, we defined a small meta-evaluation of our tool. This meta-evaluation consisted on triggering the evaluation on the command line with a small automation script, as well as opening each of the HTML test documents in the browser, and triggering the evaluation through the supplied *bookmarklet*.

Afterwards, we compared the evaluation outcome (warn/-pass/fail by technique) for all HTML test documents and compared their results with the previously defined expected results. Since all of these HTML tests do not include Javascript-based dynamics that transform their respective HTML DOM tree, we postulated that the implementation returns the same evaluation results in both evaluation environments.

5. EXPERIMENTAL STUDY

We devised an experimental study on the home pages from the Alexa Top 100 Web sites[4]. This study centred on analysing how Web accessibility evaluation results in different outcomes for the *Command Line* and *Browser* environments.

Next, we detail the setup of this experiment, followed by a description of how data was acquired and processed. Finally, we present the most significant results from our experiment.

5.1 Setup

We started by checking if each Web site could be reached, and if we got an HTTP response with its corresponding home page. In one of the cases, the domain is being used for serving ancillary resources for other Web sites. Other Web sites were also unavailable, for unknown reasons. Finally, we filtered the Web sites that were blocked from the university network (mostly illegal file sharing or adult content services).

The resulting set of Web sites that were to be evaluated comprises a total of 82 reachable home pages.

[4]Alexa Top 100: `http://http://www.alexa.com/topsites`

5.2 Data Acquisition and Processing

We accessed the Web pages and saved the original HTML documents (through the *command line* environment) and the transformed HTML documents (through the *browser* environment), so we could repeat the assessments with these documents, if necessary. We performed the evaluations in both environments sequentially to the same Web page, and with little temporal differences. This way we avoided the potential content differences between the HTTP responses in both environments, which could lead to incorrect evaluation results. The resulting time delta between the evaluations in both environments averages at 89.72 seconds, $\sigma = 69.59$.

In some cases on the browser environment, we were faced with strong safeguards that deflected our ability to inject our evaluation procedures into the HTML document (often implemented as safeguards for cross-site scripting attacks). For these cases, we eliminated these restrictions and successfully evaluated the documents afterwards.

On browser's partial fixing of HTML, we want to take that into account in the comparison of evaluation environments, since users are faced with the fixed content.

Finally, with all evaluations finished, we transformed all EARL results into corresponding CSV format for subsequent analysis, as detailed in the implementation Section.

Our evaluation yielded differences in the size of the HTML documents, both in terms of absolute bytes and HTML elements, when comparing these numbers between evaluation environments. The average difference on the byte size of the documents is 2885 bytes, $\sigma = 51181.63$, which supports the idea that Web pages can have several transformations in their content between environments. In terms of HTML element count, there is an average difference of 72.5 elements, $\sigma = 693.56$. These results indicate that, in fact, there are differences in the HTML between these two environments.

We investigated further these numbers, in order to understand if there were any cases where the size of the documents, in bytes and number of HTML elements, increase or decrease in absolute values. These results are depicted in Figures 6 and 7, respectively.

In terms of absolute byte size for the evaluated Web pages, the *command line* environment yields an average of 69794 bytes, $\sigma = 95358.67$, while averaging at 81007.02 bytes in the *browser* environment, $\sigma = 126847.75$. This scenario repeats for HTML elements, where the *command line* clocks at 915.71 elements on average, $\sigma = 1152.11$, and 1154.72 elements on average for the *browser* environment, $\sigma = 1565.87$.

This outcome reflects the underlaying assumption made in the hypothesis, i.e., that the difference between HTML documents in both environments is real, and very significative. Based on this, we present in the next Section an analysis on how accessibility evaluation – based on WCAG 2.0 – becomes evident on the *command line* and *browser* environments.

5.3 Results

We focused our study in two main set of results: first, the difference of evaluation outcomes (*fail*, *pass*, *warning*) between both environments; and second, what outstanding Web accessibility evaluation criteria are able to characterise the differences between evaluating in each environment. The next Sections detail our corresponding findings.

5.3.1 Evaluation Outcomes

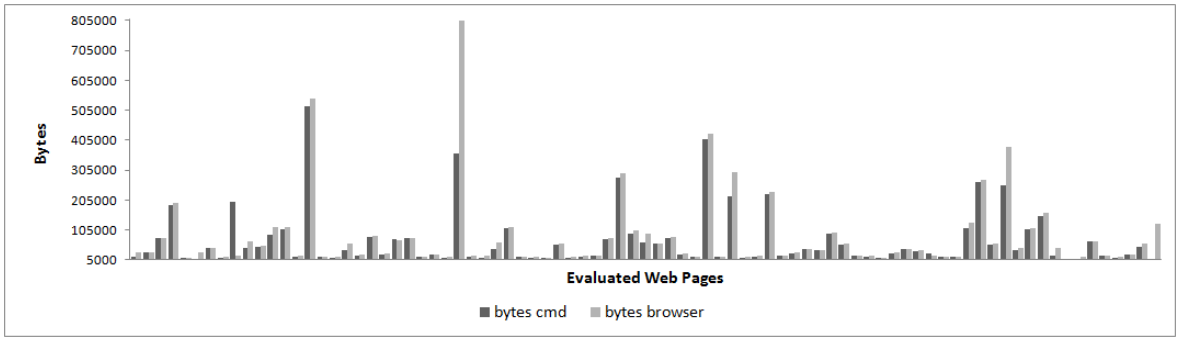

Figure 6: Comparing size in bytes in both environments

We have detected that there are significant differences in the number of HTML elements detected by by Web accessibility evaluation procedures between both environments. In Figures 8, 9, and 10 we present how the three evaluation outcomes (*fail*, *pass*, *warn*, respectively) differ between environments. A *failure* occurs in the cases where the evaluator can automatically and unambiguously detect if a given HTML element has an accessibility problem, whereas the *passing* represents its opposite. *Warnings* are raised when the evaluator can partially detect accessibility problems, but which might require additional inspection (often by experts).

Inspecting these results with additional detail, the Web pages have the following evaluation outcomes:

- *Pass*: an average 9.67 elements pass their respective evaluation criteria (σ = 19.12) in the command line environment. However, this number highly increases in the browser environment to an average 272.78 elements (σ = 297.10), ie, 46%;
- *Fail*: an average 47.44 elements fail their respective evaluation criteria (σ = 70.82) in the command line environment. This number increases in the browser environment to an average 90.10 elements (σ = 125.93), ie, 12%;
- *Warn*: an average 425.02 elements pass their respective evaluation criteria (σ = 682.53) in the command line environment. This number increases in the browser environment to an average 685.21 elements (σ = 1078.10), ie, 45%.

Next, we detail how evaluation criteria differentiate between both evaluation environments.

5.3.2 Evaluation Criteria

WCAG 2.0 defines a set of evaluation criteria for each of its general accessibility guidelines. Our experimental study resulted in several interesting outcomes from the accessibility evaluation. As it can be grasped from Figure 11 (log-scale on HTML Elements count), each one of the implemented criteria is invariantly applied more times in the browser environment than in the command line environment.

However, these results still mask an important detail about criterion applicability: there might be Web pages where any given criterion could be applied in the command line environment, but dismissed in the browser environment (i.e., *false positives*). Likewise, the opposite situation can also

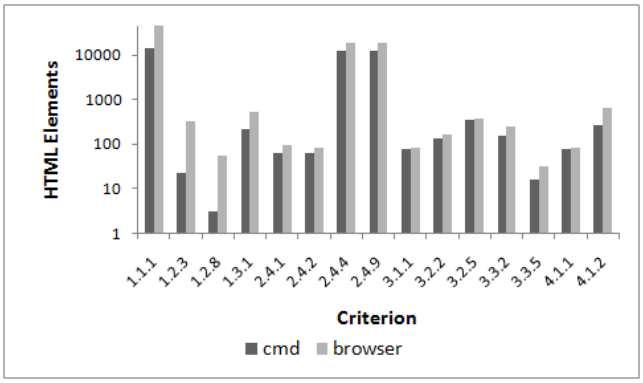

Figure 11: Browser vs Command Line per criterion (log-scale on HTML Elements count)

arise (i.e., *false negatives*). In other words, *false negatives* and *false positives* occur due to the differences between evaluation results of both environments, for instance, failing on Criterion 1.1 (i.e., alternative texts) in command line evaluation, but passing in the browser (e.g., a script introduced alternative texts for images). This is a false negative yield by command line evaluation, since users are faced with its browser counterpart.

Consequently, in this analysis, we discovered some cases where specific criteria in fact resulted in both *false positives* and *false negatives*, when using the command line environment results as the baseline for comparison. This resulted in the outcomes depicted in Table 1.

This analysis shows that, in fact, nearly 67% of the cases (10 criteria out of the 15 that were implemented) in the command line environment yield false negatives, i.e., were unable to be applied. The occurrence of false positives, i.e., when a Web page version for the command line environment triggered the application of criteria but not on the browser environment, was substantially lower, though.

Next, we delve into four WCAG 2.0 criteria that reflect the different evaluation natures that emerge from the comparison of the outcomes from the two evaluation environments: *1.1.1*, *1.2.3*, *2.4.4*, and *3.1.1*.

5.3.2.1 WCAG 2.0 Criterion 1.1.1.

Criterion 1.1.1 is the *poster child* of Web accessibility adequacy (both in engineering and evaluation terms). It reflects

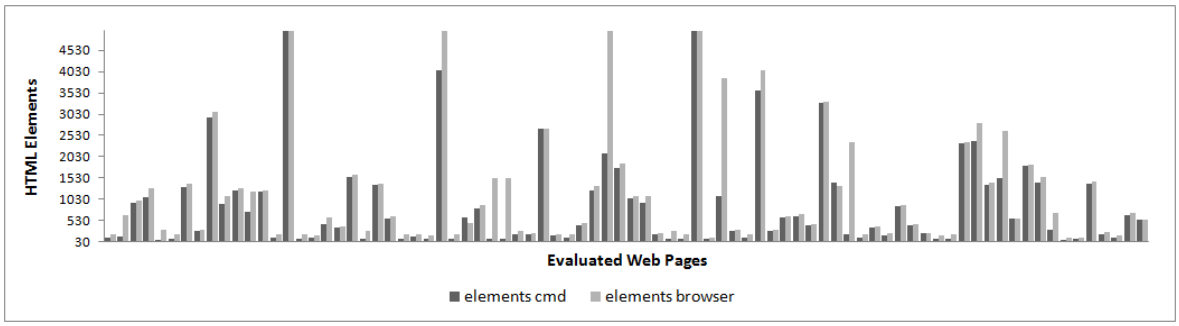

Figure 7: Comparing size in HTML Elements count in both environments

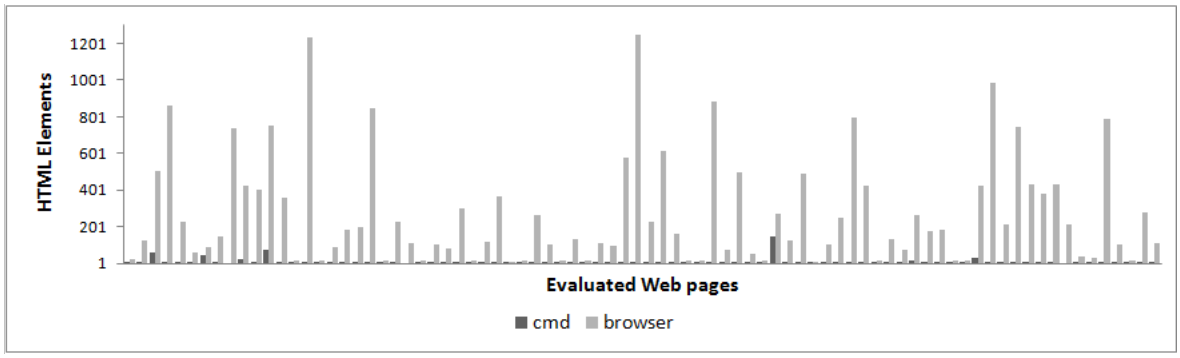

Figure 8: Number of HTML Elements that Passed

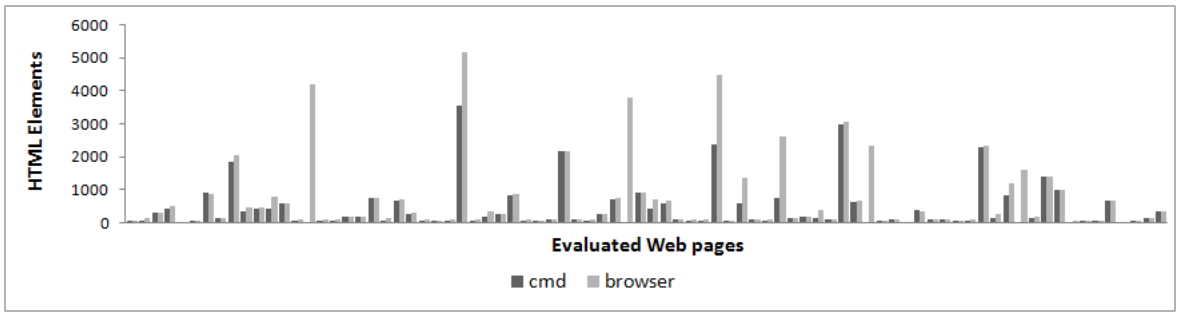

Figure 9: Number of HTML Elements that Failed

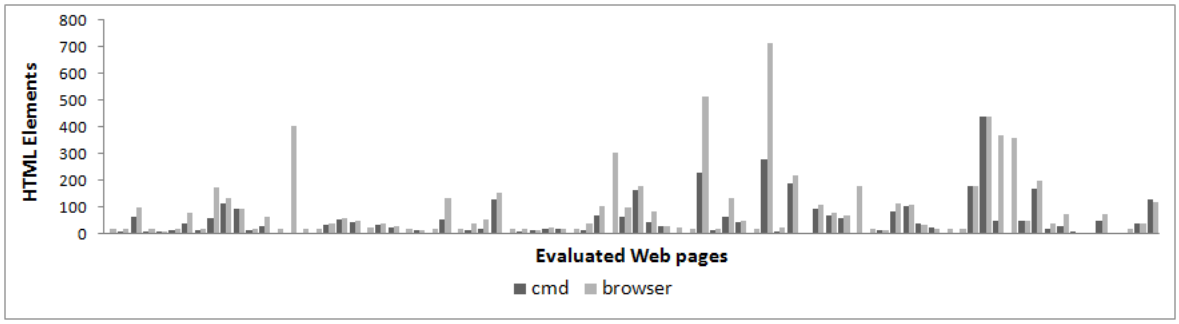

Figure 10: Number of HTML Elements that had Warnings

Table 1: False positives and false negatives in criteria applicability on *command line* environment

Criterion	False positives	False negatives
1.2.3		11%
1.2.8	2%	12%
1.3.1		27%
3.1.1		6%
3.2.2		9%
3.2.5	1%	5%
3.3.2		9%
3.3.5		6%
4.1.1		1%
4.1.2		37%

the necessity for content equivalence, thus enabling content understanding no matter what impairment a user has. For instance, the existence of alternative textual descriptions for images. Thus, we analysed individually this criterion, as depicted in Figure 12.

For a significant number of the Web pages we analysed, there is a high increase of situations that could be detected in the browser context. A brief glance at these differences showed the dynamic injection of images at either the *DOM Ready* or *DOM Load* browser rendering events. This kind of disparity on the results is the one that occurs more often for all of the implemented criteria.

5.3.2.2 WCAG 2.0 Criterion 1.2.3.

Criterion 1.2.3 depicts, in Figure 13, one case of the aforementioned false negatives. Almost all of the detected applicability occurred in the browser environment.

5.3.2.3 WCAG 2.0 Criterion 2.4.4.

In the case of Criterion 2.4.4, as depicted in Figure 14, most of the results are typical. However, as identified in the graph, there is a Web page where the command line environment detects a substantially bigger amount of problems for this criterion. While not all of those cases disappear in the browser environment, it shows that even when no false positive is raised for a criterion's applicability, there are cases where dynamic scripts remove detectable accessibility issues.

5.3.2.4 WCAG 2.0 Criterion 3.1.1.

Finally, Criterion 3.1.1, as depicted in Figure 15, allows for the detection of the (un)availability of form submission buttons. This could not be detected in the command line environment (i.e., the missing gaps in the graph), as these buttons were dynamically injected into the Web page.

6. DISCUSSION

Our study on the resulting outcomes from evaluating Web accessibility in the *command line* and *browser* environments has yielded an interesting amount of insights, respecting to automated Web accessibility evaluation practices. In the light of the results presented in the previous Section, we revisit the research hypothesis that initiated our study:

> *Evaluating Web content in the browser provides more accurate and more in-depth analysis of its accessibility.*

In the next Sections, we discuss how Web accessibility can be evaluated in the browser, and finish with a discussion of the limitations of our experimental setup.

6.1 Web Accessibility Evaluation in the Browser

Our expectations with regards to the raised hypothesis were confirmed. Indeed, there are deep differences in the accessibility evaluation between the *command line* and *browser* environments. This is reflected not just in the additional amount of processable HTML elements, but on the rate of false negatives and positives yielded by command line environment evaluations as well.

Hence, it is important to stress that evaluating the accessibility of modern Web pages in a *command line* environment can deliver misleading paths for designers and developers due to the following reasons:

- There are significant differences between the structure and content of Web pages in both evaluation environments. Thus, for dynamic Web pages, developers and designers can be faced with evaluation results that reflect different HTML DOM trees. This fact, on its own, can often provide confusion and result on difficulties of detecting the actual points where accessibility problems are encountered;

- False positives at the *command line* environment provide another point that can confuse designers and developers that are faced with these accessibility evaluation results, since they become invalid in the *browser* environment (e.g., corrected with the aid of Javascript libraries);

- Finally, false negatives are more critical, since a lot of potential accessibility problems are simply not detected in the *command line* environment. Consequently, an evaluation result might pass on 100% of accessibility checks, but the HTML DOM tree that is presented to end-users faces severe accessibility problems.

We believe that these results show that, in fact, it is of the most importance to evaluate the accessibility of Web pages in the environment where end-users interact with them. The often proposed methodology of building Web pages in a *progressive enhancement* fashion (where scripts insert additional content and interactivity) do guarantee neither the improvement, nor the maintenance of the accessibility quality of any given Web page.

6.2 Limitations of the Experiment

Our experiment has faced some limitations, both in terms of its setup, as well as on the type of results that can be extrapolated, including:

- *Data gathering*: since we gathered all Web pages in the two environments at different instants, we could not guarantee 100% that Web page generation artefacts were not introduced between requests for each of the evaluated Web pages. Furthermore, the presented results are valid for the sample set of Web pages that were selected. However, we believe that these pages are representative of modern Web design and development of front-ends;

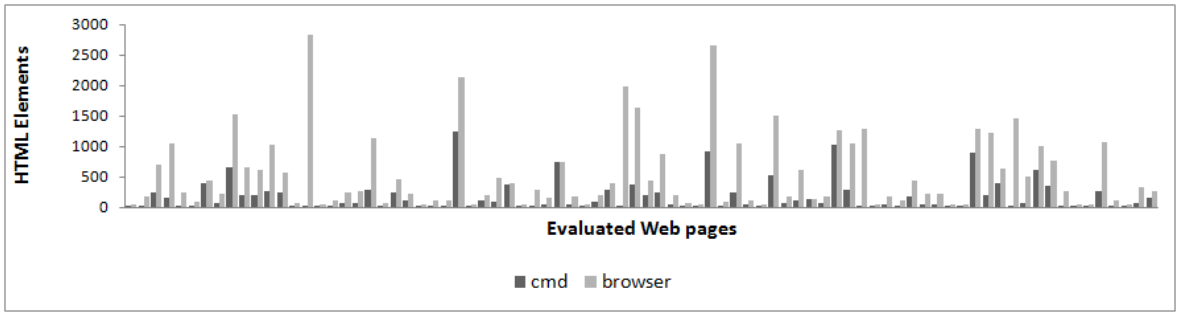

Figure 12: Browser vs Command Line for criterion 1.1.1

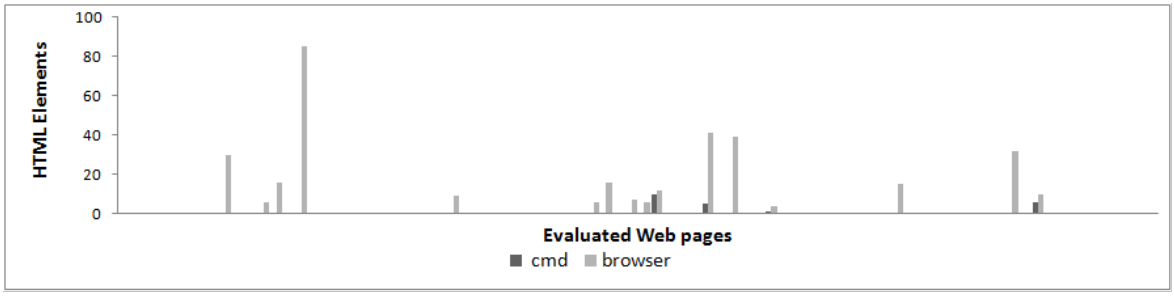

Figure 13: Browser vs Command Line for criterion 1.2.3

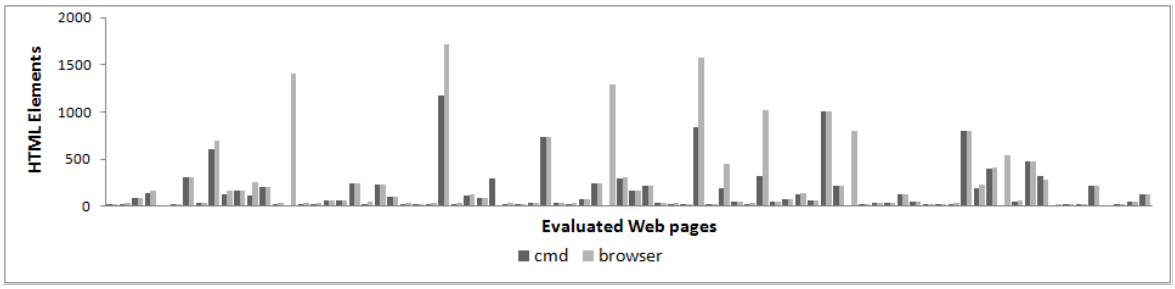

Figure 14: Browser vs Command Line for criterion 2.4.4

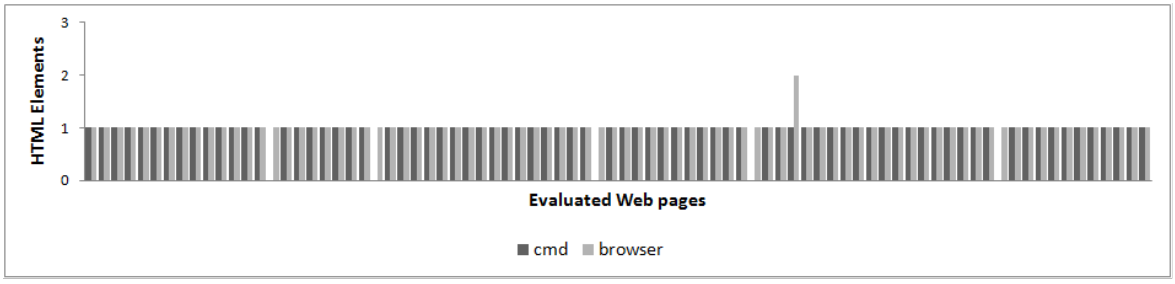

Figure 15: Browser vs Command Line for criterion 3.1.1

- *DOM trees*: while the QualWeb evaluator takes a DOM representation of the HTML, we only analysed the profusion of Web accessibility inadequacies in term of HTML elements, leaving out other potential factors that influence the accessibility of Web pages (e.g., CSS), and we did not save iFrames in the Web pages, but ultimately did not influence the evaluation because we do not look to their content;
- *Comparison of DOM trees*: our experimental setup did not provide enough information to pinpoint what transformations to the HTML DOM were made at both *DOM Ready* and *DOM Load* phases;
- *Script injection*: we encountered some cases (notably, `facebook.com`) where the injection of accessibility evaluation scripts was blocked with *cross-site scripting* (XSS) dismissal techniques. In these cases, we hand crafted minimal alterations on these Web pages, in order to disable these protections. Nevertheless, none of these alterations influenced the outcome of the accessibility evaluations performed in these cases;
- *Automated evaluation*: since this experiment is centred on automated evaluation of Web accessibility quality, it shares all of the inherent pitfalls. This includes the limited implementation coverage of WCAG 2.0.

7. CONCLUSIONS AND FUTURE WORK

This paper presented an experimental study of automated Web accessibility evaluation in the context of two environments: *command line* and *browser*. For this experiment, we analysed the accessibility quality of the home pages of the 100 most visited Web sites in the world. We provided evidence that the significant differences introduced by AJAX and other dynamic scripting features of modern Web pages do influence the outcome of Web accessibility evaluation practices. We showed that automated Web accessibility evaluation in the *command line* environment can yield incorrect results, especially on the applicability of success criteria.

Facing with the obtained results, and based on the implementation of the *QualWeb evaluator* and environment evaluation framework, ongoing work is being conducted in the following directions: (1) Implementation of more WCAG 2.0 tests based on the analysis of CSS, especially in the post-cascading phase, when all styling properties have been computed by the Web browser; (2) Continuous monitoring of changes in the HTML DOM, thus opening the way for detection of more complex accessibility issues, such as WAI ARIA live regions [5]; (3) Detecting the differences in DOM manipulation, in order to understand the typical actions performed by scripting in the browser context; (4) The implementation of additional evaluation environments, such as developer extensions for Web browsers (e.g., Firebug[5]), as well as supporting an interactive analysis of evaluation results embedded on the Web pages themselves.

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, the Multiannual Funding Programme, and POSC/EU.

[5]Firebug: `http://getfirebug.com/`

9. REFERENCES

[1] S. Abou-Zahra. Complete list of web accessibility evaluation tools, 2006. Last accessed on February 11th, 2011, from `http://www.w3.org/WAI/ER/tools/complete`.

[2] S. Abou-Zahra. Wai: Strategies, guidelines, resources to make the web accessible to people with disabilities - conformance evaluation of web sites for accessibility, 2010. Last accessed on November 11th, 2010, from `http://www.w3.org/WAI/eval/conformance.html`.

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

[4] M. Cooper, G. Loretta Guarino 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/`.

[5] J. Craig and M. Cooper. Accessible rich internet applications (wai-aria) 1.0. W3C working draft, W3C, Sept. 2010. `http://www.w3.org/TR/wai-aria/`.

[6] K. Ford, J. Richards, J. Allan, and J. Spellman. User agent accessibility guidelines (UAAG) 2.0. W3C working draft, W3C, July 2009. `http://www.w3.org/TR/2009/WD-UAAG20-20090723/`.

[7] 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)*, New York, NY, USA, 2009. ACM.

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

[9] I. Jacobs and N. Walsh. Architecture of the World Wide Web, Volume One. W3C Recommendation, World Wide Web Consortium (W3C), Dec 2004. Last accessed on November 9th, 2010, from `http://www.w3.org/TR/webarch/`.

[10] R. Lopes and L. Carriço. Macroscopic characterisations of Web accessibility. *New Review of Hypermedia and Multimedia*, 16(3):221–243, 2010.

[11] R. Lopes, D. Gomes, and L. Carriço. Web not for all: A large scale study of web accessibility. In *W4A: 7th ACM International Cross-Disciplinary Conference on Web Accessibility*, Raleigh, North Carolina, USA, April 2010. ACM.

[12] 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*, New York, USA, 2000. ACM.

[13] E. Velleman, C. Meerveld, C. Strobbe, J. Koch, C. A. Velasco, M. Snaprud, and A. Nietzio. Unified Web Evaluation Methodology (UWEM 1.2), 2007.

[14] 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.

A.3 An Architecture for Multiple Web accessibility Evaluation Environments - HCII 2011

An Architecture for Multiple Web Accessibility Evaluation Environments

Nádia Fernandes, Rui Lopes, Luís Carriço

LaSIGE, University of Lisbon, Edifício C6 Piso 3
Campo Grande, 1749 - 016 Lisboa, Portugal

{nadia.fernandes, rlopes, lmc}@di.fc.ul.pt

Abstract. Modern Web sites leverage several techniques that allow for the injection of new content into their Web pages (e.g., AJAX), as well as manipulation of the HTML DOM tree. This has the consequence that the Web pages that are presented to users (i.e., browser environment) are different from the original structure and content that is transmitted through HTTP communication (i.e., command line environment). This poses a series of challenges for Web accessibility evaluation, especially on automated evaluation software.
In this paper, we present an evaluation framework for performing Web accessibility evaluations in different environments, with the goal of understanding how similar or distinct these environments can be, in terms of their web accessibility quality.

Keywords: Web Accessibility, Web Accessibility Evaluation Environments

1 Introduction

The Web, as an open platform for information production and consumption, is being used by all types of people, with miscellaneous capabilities, including those with special needs. Consequently, Web sites should be designed so that information can be perceived by everyone in the same way, i.e., should be accessible. To analyse if a given Web page is accessible it is necessary to inspect its front-end technologies (e.g. HTML, CSS, Javascript) according to specific evaluation rules. From the different ways this inspection can be done, an interesting evaluation procedure concerns the usage of accessibility assessment software tools that algorithmically inspect a Web page's structure and content in an automated way.

Automatic accessibility evaluation can be performed in original or transformed HTML, resulting in different environments on which assessment takes place. One of the environments concerns the original HTML which is the HTML document derived from the HTTP. The other environment concerns the transformed HTML which is the resulting application of front-end technologies into the original HTML, as processed by CSS and AJAX/Javascript. This can substantially change the content struc-

ture, presentation, and interaction capabilities provided by a given Web page. This distinction between the original and transformed versions of a Web page's HTML is critical, since it is the latter that is presented and interacted by all users within a Web browser. Usually, the existent automatic evaluation procedures, such as those presented in [5, 10, 11], occur in the original HTML.

This paper presents an evaluation framework for perform Web accessibly evaluations in different environments. Taking into account that usually the existents automatic evaluation procedures occur in the original HTML conclusions over the accessibility quality of a Web page can be incomplete, or, in extreme erroneous. It is therefore important to access the transformed HTML documents and understand how deep the differences toward the original document are.

2 Related Work

To help create accessible Web pages, the Web Accessibility Initiative (WAI) developed a set of accessibility guidelines, the Web Content Accessibility Guidelines (WCAG) [9], that encourage 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 [1, 2]. Although these guidelines exist and are supposed to be followed by the creators, most Web sites still have accessibility barriers that make very difficult or even impossible many people to use them [1]. Thus, WCAG can also be used as a benchmark for analysing the accessibility quality of a given Web page.

Web Accessibility Evaluation is an assessment procedure to analyse how well the Web can be used by people with different levels of disabilities, as detailed in [1]. Optimal results are achieved with combinations of the different approaches of Web accessibility evaluation, taking advantage of the specific benefits of each of them [1]. Therefore, conformance checking [3], e.g., with the aid of automated Web accessibility evaluation tools is an important step for the accessibility evaluation.

Automated evaluation is performed by software, i.e., it is carried out without the need of human intervention, which has the benefit of objectivity [2]. To verify where and why a Web page is not accessible it is important to analyse the different resources that compose the Web page. Two examples of automatic accessibility evaluators are: EvalAcess [6] that produces a quantitative accessibility metrics from its reports and the automatic tests of UWEM [5].

In the past, the predominant technologies in the Web were HTML and CSS, which resulted in *static* Web pages. Today, on top of these technologies, newer technologies appear (e.g., Javascript), and, consequently, the Web is becoming more and more dynamic. Nowadays, user actions and/or automatically triggered events can alter a Web page's content. Because of that, the presented content can be different from the initially received by the Web browser. To solve this problem, the accessibility evaluation should be applied to new environments, i.e., in the Web browser context. However, automatic evaluations do not consider these changes in the HTML document and, because of that, results can be wrong and/or incomplete. Expert and user evaluation are performed in the browser, they do not suffer with these changes.

The importance of the Web browser context in the evaluation results is starting to be considered and is already used in three tools named *Foxability*, *Mozilla/Firefox Accessibility Extension*, and *WAVE Firefox toolbar* [7]. However, these tools focus only evaluating Web pages according to WCAG 1.0. Furthermore, since their evaluation procedures are embedded as extensions, they become more limited in terms of their application.

Also, since these tools focus on providing developer-aid on fixing accessibility problems, the resulting outcomes from evaluations are user-friendly, thus less machine-friendly. Moreover, this "browser paradigm" - like is called in [7] - is very preliminary. Until now, to the best of our knowledge, differences between results in different evaluation environments are not clear. To perform correct comparisons, it must be guaranteed that tests are implemented in different environments in the same way, by reducing implementation bias.

3 Web Accessibility Evaluation Environments

Our study is emphasized in two main environments: Command Line and Browser. The Command Line environment represents the typical environment for automated evaluation, which includes existing evaluators that can be accessed online and the evaluation is performed into the original HTML document. In Browser environment users interact with the Web evaluation, performed into the transformed version of the HTML document.

Consequently, to better grasp the differences between the environments, we defined an architecture that allows for leveraging the same evaluation procedures in any environment, as detailed below. Afterwards, we explain how we implemented the ideas from this architecture, as well as how it was validated.

3.1 Architecture

The architecture of our evaluation framework is composed by five components, as depict in Figure 1: the QualWeb Evaluator, the Environments, the Techniques, the Formatters and the Web Server.

The *QualWeb Evaluator* is responsible for performing the accessibility evaluation in Web pages using the capabilities provided by the *Techniques* component; it uses the *Formatter* component to tailor the results into specific serialisation formats, such as EARL reporting [8]. Finally, *QualWeb Evaluator* can also be used in different *Environments*.

The *Techniques* component contains the individual front-end inspection code that is intended to be used in evaluation. In our case we chose the WCAG 2.0 [9], because it is one of most important accessibility standards. The *Techniques* component is built so that other techniques could be added, at any time, to be used in the evaluator.

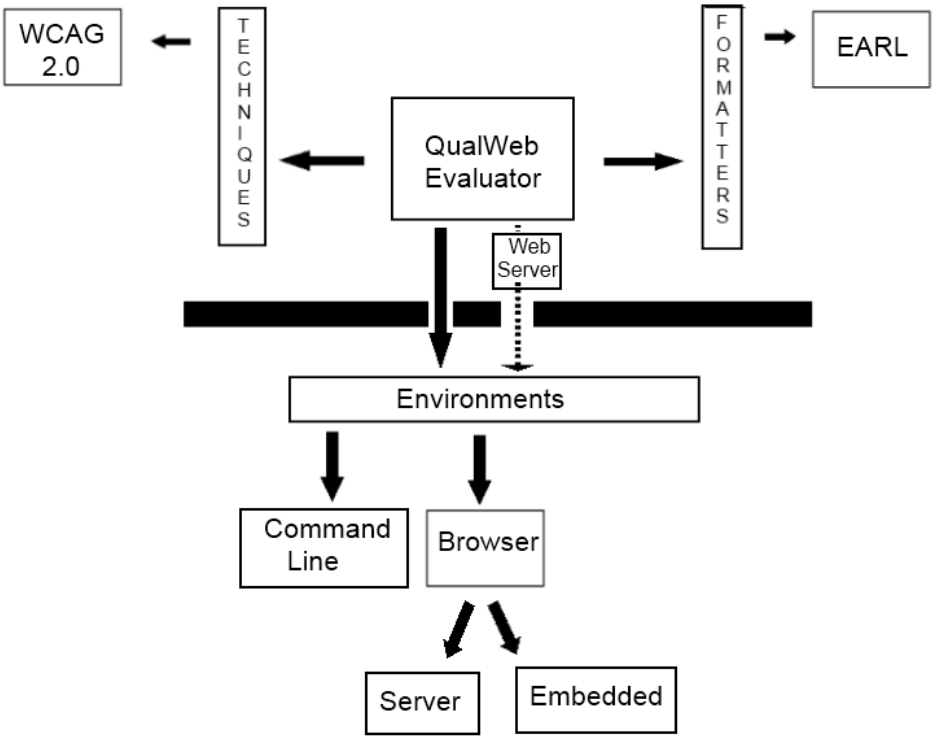

Fig. 1. Architecture of the Evaluation Framework.

The *Browser* is the environment where the transformed HTML is used and the evaluation is performed in a browser. In *Browser* could be consider two mechanisms to deliver the evaluation results, the *Server* and the *Embedded*. In the *Server* the HTML document is evaluated and the result is sent to the Web Server for subsequent analysis. In the *Embedded* the evaluation results are injected into the HTML document and shown to the developers/designers directly within the Web page.

Furthermore, other environments can be added to Environments component, in order to supply different HTML representations.

3.2 Implementation

In order to compare the proposed evaluation environments, we must use the same accessibility evaluation implementation. Given that one of the environments is the Web browser, we have a restriction on using Javascript as the implementation language. Thus, to develop the Command Line version of the evaluation process, we

leveraged Node.js[1] an event I/O framework based on the V8 Javascript engine[2]. In addition to standard Node.js modules, we used several other ancillary modules[3], including:

- *Node-Static*, which allowed for serving static files into the browser environment;
- *Node-Router*, a module that supports the development of dynamic behaviours, which we used to implement the retrieval and processing of evaluation results, and
- *HTML-Parser*, which provides support for building HTML DOM trees in any environment.

Besides these standard modules, we also implemented a set of modules for our evaluation framework, including:

- *EARL* module, which allows for the creation of EARL documents with the defined templates and parse EARL files using the *Libxmljs* library, and
- *Evaluator* module, which performs the accessibility evaluation with the implemented techniques.

Next it is presented an excerpt from WCAG 2.0 H64 technique.

```
function inspect(DOMList)
{
  if (typeof DOMList == "undefined" || DOMList.length
== 0)
    return;

  for (var i = 0; i < DOMList.length; i++)
{
    position++;
    if (DOMList[i]["type"] == "tag" && (DOML
       ist[i]["name"] == "frame" || DOMList[i]["name"]
==
       "iframe"))
{
      if(DOMList[i]["attribs"]["title"] != "" && DOML
        ist[i]["attribs"]["title"] != "undefined" &&
        DOMList[i]["attribs"]["title"] != "'" )
      {
        addElement(position,'cannotTell: title could not
        describe frame or frame',"");
      }
```

[1] Node.js: http://nodejs.org/
[2] V8 Javascript engine: http://code.google.com/p/v8/
[3] GitHub modules: https://github.com/ry/node/wiki/modules/

```
        else
            addElement(position,'failed',"");
      }
      inspect(DOMList[i]["children"]);
    }
  }
  exports.startEvaluation=startEvaluation;
```

Next, we present additional details on how we implemented both evaluation environments, as well as report generation and processing capabilities.

3.2.1 Command Line Environment

This environment obtains the HTML document from a URL using an HTTP request, executes the QualWeb evaluator on the HTML DOM tree, and serialises its outcome into EARL. All of these processes are implemented with a combination of the HTML-Parser, EARL, and Evaluator modules, executed from a command line.

3.2.2 Browser Environment

This environment uses a *bookmarklet* (Figure 2) to trigger the execution of the evaluation within the browser. Bookmarklets are a kind of browser bookmark that has the particularity of point to a URI that starts with the `javascript`: protocol. In front of this, pure Javascript commands follow. Thus, when a user activates the bookmarklet, these commands are executed.

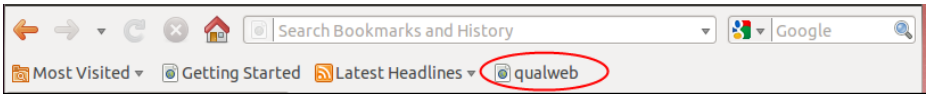

Fig. 2. Evaluation execution example on Browser.

In the case of our evaluator, this bookmarklet injects the necessary functions to obtain the HTML DOM tree of the current Web page, executes the QualWeb evaluator, and sends the evaluation results to a server component. These results are transformed in the EARL serialisation format, and subsequently stored. To implement this browser-server execution and communication mechanism, we used the following modules:

- *Bootstrap*, to import the required base modules, and
- *LAB.js*, to inject all of the evaluation modules into the browser's DOM context.

3.2.3 Report Generation and Processing

Finally, to generate the evaluation reports containing the accessibility quality results, we used the following modules:

- *Node-Template*, to define EARL reporting templates,
- *Libxmljs*, to parse EARL reports, and
- *CSV* module, to recreate a comma-separated-values (CSV) counterpart from a given EARL report. This module allowed for a better inspection and statistical analysis with off-the-shelf spreadsheet software. Besides, to the best of our knowledge, there was nothing that performs the EARL parsing giving results in CSV.

While the EARL format allows for the specification of evaluation results, we had to extend EARL with a small set of elements that could allow for the analysis of the resulting outcomes from our experiment. Hence, we defined a Metadata field that supports the specification of HTML element count, as well as a Timestamp to state the specific time when the evaluation was performed.

The EARL reports served as the basis for generating CSV reports. Due to the extensiveness of EARL reports generated by our evaluator, especially in what respects to parsing and consequent memory consumption provided by generic DOM parsers, we implemented the EARL-CSV transformation procedures with SAX events.

Next, an EARL document example in RDF/N3[4] format.

```
<#QualWeb> dct:description ""@en;
  dct:hasVersion "0.1";
  dct:location "http://qualweb.di.fc.ul.pt/";
  dct:title "The QualWeb WCAG 2.0 evaluator"@en;
  a earl:Software.
<assertion1> dc:date "1291630729208";
  a earl:Assertion;
  earl:assertedBy <assertor>;
  earl:mode earl:automatic;
  earl:result <result1>;
  earl:subject <http://ameblo.jp/>;
  earl:test <http://www.w3.org/TR/WCAG20-
TECHS/H25#H25>.
<http://ameblo.jp/> dct:description ""@en;
  dct:title "The QualWeb WCAG 2.0 evaluator"@en;
  qw:elementCount "381";
  a qw:metadata,
  earl:TestSubject.
<http://www.w3.org/TR/WCAG20-TECHS/H25> dct:hasPart
<http://www.w3.org/TR/WCAG20-TECHS/H25#H25-tests/>;
  dct:isPartOf <http://www.w3.org/TR/WCAG20-TECHS/>;
  dct:title "H25"@en;
  a earl:TestCase.
<QualWeb> dct:description ""@en;
```

[4] RDF/N3: http://www.w3.org/DesignIssues/Notation3

```
    dct:hasVersion "0.1";
    dct:title "The QualWeb WCAG 2.0 evalua-tor"@en;
    a earl:Software;
    foaf:homepage qw:.
  <result1> dct:description "descrip-
  tion"^^rdf:XMLLiteral;
    dct:title "Markup Valid"@en;
    a earl:TestResult;
    earl:info "info"^^rdf:XMLLiteral;
    earl:outcome earl:passed;
    earl:pointer <1>.
```

3.3 Testability and Validation

We developed a test bed comprising a total of 102 HTML documents, in order to verify if all the WCAG 2.0 implemented techniques provide the expected results. Each HTML document was carefully hand crafted and peer-reviewed within our research team, in order to guarantee a high level of confidence on the truthfulness of our implementation. For each technique success or failure cases were performed to test all the possible techniques outcomes. To get a better perspective on the implementation of our tests, we leveraged the examples of success or failure cases described for each WCAG 2.0 technique. The graph depicted in Figure 3 shows the number of HTML test documents defined for each technique that was implemented in the QualWeb evaluator.

To test the proper application of the implemented techniques in the two evaluation environments, we defined a small meta-evaluation of our tool. This meta-evaluation consisted on triggering the evaluation on the command line with a small automation script, as well as opening each of the HTML test documents in the browser, and triggering the evaluation through the supplied bookmarklet.

Afterwards, we compared the evaluation outcome for all HTML test documents and compared their results with the previously defined expected results. Since all of these HTML tests do not include Javascript-based dynamics that transform their respective HTML DOM tree, we postulated that the implementation returns the same evaluation results in both evaluation environments.

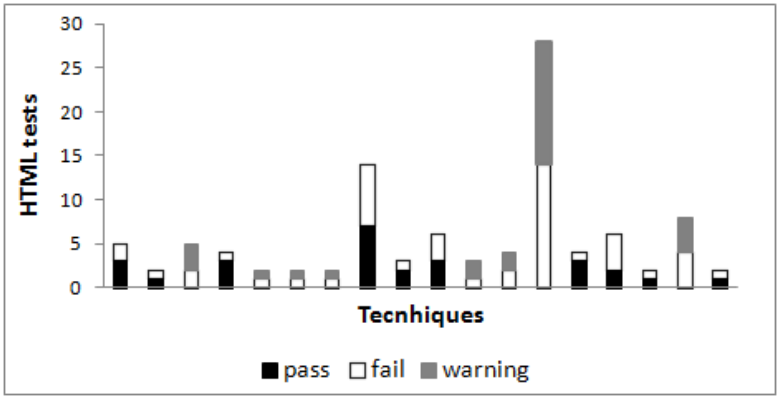

Fig. 3. Number of Test Documents per Technique.

4 Conclusions and Future Work

The presented architecture for Multiple Web Accessibility Evaluation Environments that was implemented for: *Command Line* and *Browser* environments. The architecture was used in accessibility evaluation tests successfully. In this work were implemented new modules to facilitate this type of evaluations. These modules will be available online.

Some limitations of this work are: the evaluations do not occur exactly at the same time in both environments, so we could not guarantee 100% that Web page generation artefacts were not introduced between requests for each of the evaluated Web pages, and injection of accessibility evaluation scripts could be blocked with *cross-site scripting* (XSS) dismissal techniques.

Ongoing work is being conducted in the following directions: 1) an in-depth implementation of WCAG 2.0 techniques for different front-end technologies, as well as its application in different settings and scenarios; 2) implementation of more WCAG 2.0 tests; 3) continuous monitoring of changes in the HTML DOM thus opening the way for detection of more complex accessibility issues, such as WAI ARIA live regions [12]; 4) detecting the differences in DOM manipulation, in order to understand the typical actions performed by scripting in the browser context, and 5) the implementation of additional evaluation environments, such as developer extensions for Web browsers (e.g., Firebug[5]), as well as supporting an interactive analysis of evaluation results embedded on the Web pages themselves.

Acknowledgements. This work was funded by Fundação para a Ciência e Tecnologia (FCT) through the *QualWeb* national research project PTDC/EIA-EIA/105079/2008, the Multiannual Funding Programme, and POSC/EU.

5 References

1. S. Harper and Y. Yesilada. Web Accessibility Springer, London, United Kingdom, 2008.
2. R. Lopes, D. Gomes, and L. Carriço. Web not for all: A large scale study of web accessibility. In W4A: 7th ACM International Cross-Disciplinary Conference on Web Accessibility, Raleigh, North Carolina, USA,April 2010. ACM.
3. S. Abou-Zahra. Wai: Strategies, guidelines, resource to make the web accessible to people with disabilities conformance evaluation of web sites for accessibility 2010. Last accessed on November 11th, 2010, from http://www.w3.org/WAI/eval/conformance.html.
4. 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.

[5] Firebug: http://getfirebug.com/

5. E. Velleman, C. Meerveld, C. Strobbe, J. Koch, C. A.Velasco, M. Snaprud, and A. Nietzio. Unified Web Evaluation Methodology (UWEM 1.2), 2007.
6. 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.
7. 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
8. S. Abou-Zahra and M. Squillace. Evaluation and report language (EARL) 1.0 schema. Last call WD, W3C, Oct. 2009. http://www.w3.org/TR/2009/WD-EARL10-Schema-20091029/
9. B. Caldwell, M. Cooper, W. Chisholm, L. Reid, and G. Vanderheiden, Web Content Accessibility Guidelines 2.0. , 2008. , W3C Recommendation, World Wide Web Consortium (W3C) http://www.w3.org/TR/WCAG20/
10. T. Sullivan, and R.Matson, 2000. Barriers to use: usability and content accessibility on the Web's most popular sites. CUU '00: Proceedings of the Conference on Universal Usability. New York, NY, USA: ACM, 139–144.
11. M. Vigo, M. Arrue, G. Brajnik, R. Lomuscio, and J. Abascal, 2007. Quantitative metrics for measuring web accessibility. W4A '07: Proceedings of the 2007 international cross-disciplinary conference on Web accessibility (W4A). New York,NY, USA: ACM, 99–107.
12. J. Craig and M. Cooper. Accessible rich internet applications (wai-aria) 1.0. W3C working draft, W3C, Sept. 2010. http://www.w3.org/TR/wai-aria/.

A.4 The Role of Templates on Web Accessibility Evaluation - Assets 2011

The Role of Templates on Web Accessibility Evaluation

Nádia Fernandes, Rui Lopes, Luís Carriço
LaSIGE/University of Lisbon
Campo Grande, Edifício C6
1749-016 Lisboa, Portugal
{nadiaf,rlopes,lmc}@di.fc.ul.pt

ABSTRACT

This paper presents an experimental study designed to understand the impact of HTML template usage in accessibility evaluation reporting. Our study shows that, in average, about 39% of the accessibility evaluation results for each page on a Web site are applicable to page common elements and thus are reported at least twice. This number distorts the development team's perception of the Web site corrective effort, unnecessarily hindering the distribution of work.

Categories and Subject Descriptors

H.5.4 [**Information Interfaces and Presentation**]: Hypertext/Hypermedia—*User issues*; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Evaluation/methodology*; K.4.2 [**Computers and Society**]: Social Issues—*Assistive technologies for persons with disabilities*

General Terms

Measurement, Human Factors.

Keywords

Web Accessibility, Templates, Automated Evaluation.

1. INTRODUCTION

Front-end Web development is highly centred on the use of templates to ease implementing and maintaining coherence of Web site structural features. An estimate 40-50% of Web content uses templates [1]. However, automatic accessibility evaluations are usually done in pages as a whole, i.e., after all templates are composed into the Web page's final form.

As such, evaluating Web sites could lead to misleading accessibility evaluation results, i.e., the same errors are repeated over and over obfuscating the final reports. This exacerbates the repairing problems, when they occur in a template, and dilute the remanding ones within the numerous reported errors. While managing repairing processes, this may simply kill the corrective project (too demanding) or difficult the distribution of correction tasks (several programmers correcting the same problem).

With template-aware accessibility evaluation tools, developer teams can better manage the accessibility repair process and have a more realistic perspective of the actual effort necessary to do it. Solutions, like doing evaluation in the original template and sources, yields heavily distorted results [3] and are not a reasonable alternative.

In order to effectively evaluate accessibility considering templates, one should first assess if the amount of errors found in common elements amongst Web pages is relevant in respect to the total amount. Although this does not conclude the work, it is a fundamental contribution that is addressed in this paper.

We propose the use of a simple algorithm to identify common elements amongst the HTML DOM trees. This will only provide an approximation of the template elements used in its construction, but offers a reasonable estimate for initial assessment. On the other hand, it will also raise the developers' awareness to other common elements, not contained in templates that could be addressed in the corrective processes. We conducted a preliminary study that demonstrates that a significant part the accessibility errors found in relevant Web pages occur in common elements.

2. EXPERIMENTAL STUDY

This study centred on analysing similarities of HTML elements between Web pages. The similarity criteria targets typical template-based definitions. We used the *Fast Match* algorithm [2] to obtain a measure of similarity, and applied it on pairs of HTML DOM trees.

The study was performed in a set of sites that feature a consistent use of HTML. The selection rationale was to select well-known and representative Web sites from the Alexa Top 100 Web sites[1] – *Google*, *Wikipedia*, *Facebook* and *Amazon* –, two modern online Portuguese newspapers – *DN* and *Público* – and the *WordTaps*. *WordTaps* uses *WordPress* and it is a well-known blogging and Web site platform.

We selected a Web page of each Web site, other than the home page. For each Web page, we compare it with the home page, to obtain the set of elements that are common between them (the *template* set), and the set that is specific for the Web page (the *specific* set). Each Web page is then assessed using the automatic QualWeb evaluator [3] and the reported errors are matched with the elements in

[1] Alexa Top 100: http://www.alexa.com/topsites

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...$5.00.

the abovementioned sets. This division allows faster access to each type of accessibility evaluation results. The process was repeated for all the Web sites.

2.1 Results

We focused our study on the percentage of WCAG 2.0 techniques applicability (i.e., specific outcomes - pass, warn, fail). The average of all the template sets is 38.85% (σ = 7.48), and in the specific content is 61.15% (σ = 7.48). Besides, the averages for the outcomes considered in the applicability are: 34.50% of warnings (σ = 7.00), 0.80% of fails (σ = 1.00), and 3.56% of pass (σ = 2.64). The percentage of errors that need to be repaired in the templates have an average of 38.06% (σ = 7.78). Figure 1 shows a sample of the evaluation results for the selected Web sites.

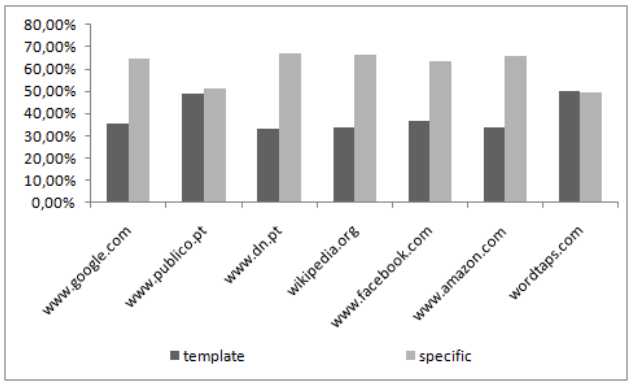

Figure 1: Applicability of WCAG 2.0 techniques on one of the evaluated Web pages.

3. DISCUSSION

These results point to a positive verification of our hypothesis: template-awareness indeed may simplify assessment reporting. We determined that approximately 39% of the results are reported at least twice, of which approximately 38% are errors that can be corrected once.

Assessment methods should be modified in order to do not consider Web pages as a whole. This way, pages can be divided, as suggested, in template and specific sets to improve evaluations and consider their various characteristics. This could be even further developed in considering elements similarities on more than two pages, and determining the number of times (more than two) that a WCAG technique is applicable to each common element on the site. As such, reporting can be additionally simplified, performance can be improved, and more accurate metrics can be defined.

Besides, regarding repairing, template aware reports can be integrated in development tools directing developers/designers to a much more effective error correction process.

4. RELATED WORK

Template detection is often used in the fields of Information Retrieval and Web page Clustering [1]. It was already refereed that templates could be considered in accessibility issues and it was suggested to use accessible content templates to preserve accessibility [5].

Accessibility results can be presented in a complex way to developers, e.g., big reports or tools that they cannot understand [4]. Therefore, reports that contain accessibility results should be simplified to facilitate developers' work, since they are not accessibility experts. If the report is self-evident, obvious and self-explanatory, to the developers, then they will understand it, without problems.

Many automatic tools generate a different instance for the same type of problem. A simplified list with the type of problem and one or two examples of the actual error is enough so that the developer can resolve the errors without major difficulties.

5. CONCLUSIONS AND FUTURE WORK

This paper presented an experimental study on detection of templates to facilitate the repair of accessibility errors. We had shown that the accessibility results of the common elements are more than a third of the whole results set. A significant percentage of the accessibility errors that would simplify error reports and consequently the developers/designers work. This way, developers/designers can repair accessibility errors only once and these are automatically repaired throughout the site.

Our experiment has some limitations and we are currently considering: 1) possible repetitions of errors for intra-page templates inside the Web page itself (e.g. list, ads); 2) explore extra templates (e.g., similar elements encoded in multiple Web pages); 3) comparing more than two pages and therefore errors that will be reported more than two times; and 4) a larger sample of sites. The Fast Match algorithm should be assessed in order to fully understand how accurately it matches the template, i.e., what elements are actually components of a template and which are not.

6. ACKNOWLEDGEMENTS

This work was funded by Fundação para a Ciência e Tecnologia (FCT) through the *QualWeb* national research project PTDC/EIA-EIA/105079/2008, the Multiannual Funding Programme, and POSC/EU.

7. REFERENCES

[1] D. Chakrabarti and R. Mehta. The paths more taken: matching dom trees to search logs for accurate webpage clustering. In *WWW '10 Proceedings of the 17th international conference on World Wide Web*, New York, NY, USA, 2010. ACM.

[2] S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom. Change detection in hierarchically structured information. In *SIGMOD '96 Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, New York, NY, USA, 1996. ACM.

[3] N. Fernandes, R. Lopes, and L. Carriço. On web accessibility evaluation environments. In *W4A '11: Proceedings of the 2009 International Cross-Disciplinary Conference on Web Accessibililty (W4A)*, New York, NY, USA, 2011. ACM.

[4] C. Law, J. Jacko, and P. Edwards. Programmer-focused website accessibility evaluations. In *Assets '05 Proceedings of the 7th international ACM SIGACCESS conference on Computers and accessibility*, New York, NY, USA, 2005. ACM.

[5] L. Moreno, P. Martinez, and B. Ruiz. Guiding accessibility issues in the design of websites. In *SIGDOC '08 Proceedings of the 26th annual ACM international conference on Design of communication*, New York, NY, USA, 2008. ACM.

Abbreviations

AT	Assistive Technology
CSS	Cascading Style Sheets
CSV	Comma-Separated Values
DOM	Document Object Model
EARL	Evaluation and Report Language
HTML	HyperText Markup Language
IDE	Integrated Development Environment
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WCAG	Web Content Accessibility Guidelines
WEA	Web Accessibility Evaluation

Bibliography

[1] A-Prompt, December 2004.

[2] A-Checker, January 2006.

[3] Foxability - Accessibility Analyzing Extension for Firefox, 2008. Last accessed on June 18th, 2011, http://foxability.sourceforge.net//.

[4] eAccessibility – Opening up the Information Society, December 2010.

[5] WAVE - Web Accessibility Evaluation Tool, 2011. Last accessed on June 18th, 2011, http://wave.webaim.org/toolbar/.

[6] Shadi Abou-Zahra. Complete List of Web Accessibility Evaluation Tools, march 2006. Last accessed on May 10th, 2011, from http://www.w3.org/WAI/ER/tools/complete.

[7] Shadi Abou-Zahra. WCAG 2.0 Test Samples Development Task Force (TSD TF) Work Statement, 2008. Last accessed on July 20th, 2010, from http://www.w3.org/WAI/ER/2006/tests/tests-tf/.

[8] Shadi Abou-Zahra. Wai: Strategies, guidelines, resources to make the web accessible to people with disabilities - conformance evaluation of web sites for accessibility, 2010. Last accessed on November 11th, 2010, from http://www.w3.org/WAI/eval/conformance.html.

[9] Shadi Abou-Zahra and Michael Squillace. Evaluation and report language (EARL) 1.0 schema. Last call WD, W3C, October 2009. http://www.w3.org/TR/2009/WD-EARL10-Schema-20091029/.

[10] Richard Atterer. Model-based automatic usability validation: a tool concept for improving web-based uis. In *NordiCHI '08 Proceedings of the 5th Nordic conference on Human-computer interaction: building bridges*, New York, NY, USA, 2008. ACM.

[11] Ziv Bar-Yossef and Sridhar Rajagopalan. Template detection via data mining and its applications. In *WWW '02 Proceedings of the 11th international conference on World Wide Web*, New York, NY, USA, 2002. ACM.

[12] Giorgio Brajnik, Yeliz Yesilada, and Simon Harper. Testability and validity of wcag 2.0: the expertise effect. In *Proceedings of the 12th international ACM SIGACCESS conference on Computers and accessibility*, ASSETS '10, pages 43–50, New York, NY, USA, 2010. ACM.

[13] Deepayan Chakrabarti and Rupesh Mehta. The paths more taken: matching dom trees to search logs for accurate webpage clustering. In *WWW '10 Proceedings of the 17th international conference on World Wide Web*, New York, NY, USA, 2010. ACM.

[14] Sudarshan Chawathe, Anand Rajaraman, Hector Garcia-Molina, and Jennifer Widom. Change detection in hierarchically structured information. In *SIGMOD '96 Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, New York, NY, USA, 1996. ACM.

[15] Wendy Chisholm, Gregg Vanderheiden, and Ian Jacobs. Web Content Accessibility Guidelines 1.0. W3C Recommendation, World Wide Web Consortium (W3C), May 1999. `http://www.w3.org/TR/WCAG10/`.

[16] Michael Cooper, Google Loretta Reid, Gregg Vanderheiden, and Ben 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/`.

[17] Michael Cooper, Loretta Guarino Reid, Gregg Vanderheiden, and Ben 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/`.

[18] Michael Cooper, Loretta Guarino Reid, Gregg Vanderheiden, and Ben Caldwell. Understanding WCAG 2.0. W3C Note, World Wide Web Consortium (W3C), October 2010. Last accessed on May 9th, 2011, from `http://www.w3.org/TR/UNDERSTANDING-WCAG20/Overview.html`.

[19] Michael Cooper, Loretta Guarino Reid, Gregg Vanderheiden, and Ben Caldwell. Understanding WCAG 2.0. W3C Note, World Wide Web Consortium (W3C), October 2010. Last accessed on May 19h, 2011, from `http://www.w3.org/TR/UNDERSTANDING-WCAG20/conformance.html`.

[20] James Craig and Michael Cooper. Accessible rich internet applications (wai-aria) 1.0. W3C working draft, W3C, September 2010. http://www.w3.org/TR/wai-aria/.

[21] José L. Fuertes, Ricardo González, Emmanuelle Gutiérrez, and Loïc 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.

[22] José L. Fuertes, Ricardo González, Emmanuelle Gutiérrez, and Loïc 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.

[23] David Gibson, Kunal Punera, and Andrew Tomkins. The volume and evolution of web page templates. In *WWW '05 Special interest tracks and posters of the 14th international conference on World Wide Web*, New York, NY, USA, 2005. ACM.

[24] Simon Harper and Yeliz Yesilada. *Web Accessibility*. Springer, London, United Kingdom, 2008.

[25] Philippe Le Hégaret. The W3C Document Object Model (DOM), 2002. Last accessed on July 20th, 2011, from http://www.w3.org/2002/07/26-dom-article.html/.

[26] Ian Hickson. HTML5 A vocabulary and associated APIs for HTML and XHTML, 2011. Last accessed on July 20th, 2011, from http://www.worldwidewebsize.com/.

[27] Ian Jacobs and Norman Walsh. Architecture of the World Wide Web, Volume One. W3C Recommendation, World Wide Web Consortium (W3C), Dec 2004. Last accessed on November 9th, 2010, from http://www.w3.org/TR/webarch/.

[28] Chris Law, Julie Jacko, and Paula Edwards. Programmer-focused website accessibility evaluations. In *Assets '05 Proceedings of the 7th international ACM SIGACCESS conference on Computers and accessibility*, New York, NY, USA, 2005. ACM.

[29] Rui Lopes and Luís Carriço. Macroscopic characterisations of Web accessibility. *Found. Trends Web Sci.*, 16(3):1–130, 20.

[30] Rui Lopes, Karel Van Isacker, and Luis 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.

[31] Rui Lopes, Karel Van Isacker, and Luís Carriço. Redefining assumptions: Accessibility and its stakeholders. In *The 12th International Conference on Computers Helping People with Special Needs (ICCHP)*. Springer, 2010.

[32] Lourdes Moreno, Paloma Martinez, and Belén Ruiz. Guiding accessibility issues in the design of websites. In *SIGDOC '08 Proceedings of the 26th annual ACM international conference on Design of communication*, New York, NY, USA, 2008. ACM.

[33] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Computing Surveys (CSUR)*, 33(1):31–88, 2001.

[34] Terry Sullivan and Rebecca 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.

[35] Eric Velleman, Colin Meerveld, Christophe Strobbe, Johannes Koch, Carlos A. Velasco, Mikael Snaprud, and Annika Nietzio. Unified Web Evaluation Methodology (UWEM 1.2), 2007.

[36] Karane Vieira, André Carvalho, Klessius Berlt, Edleno Moura, Altigran Silva, and Juliana Freire. On Finding Templates on Web Collections. *World Wide Web*, 12(2):171–211, 2009.

[37] Markel Vigo, Myriam Arrue, Giorgio Brajnik, Raffaella Lomuscio, and Julio 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.