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*

W4A2011 - Technical Paper, March 28-29, 2011, Hyderabad, India. Co-Located with the 20th International World Wide Web Conference.

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.

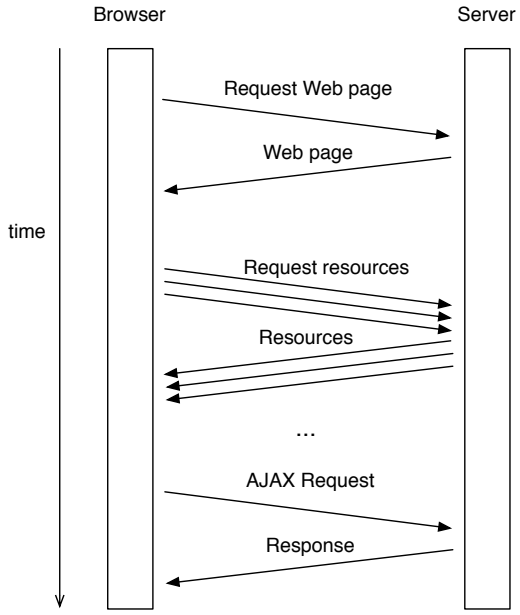

Figure 1: Web Browsing Resource Interaction

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:

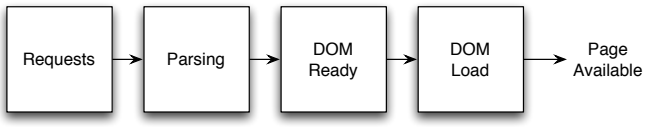

Figure 2: Web Page Loading Process

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.

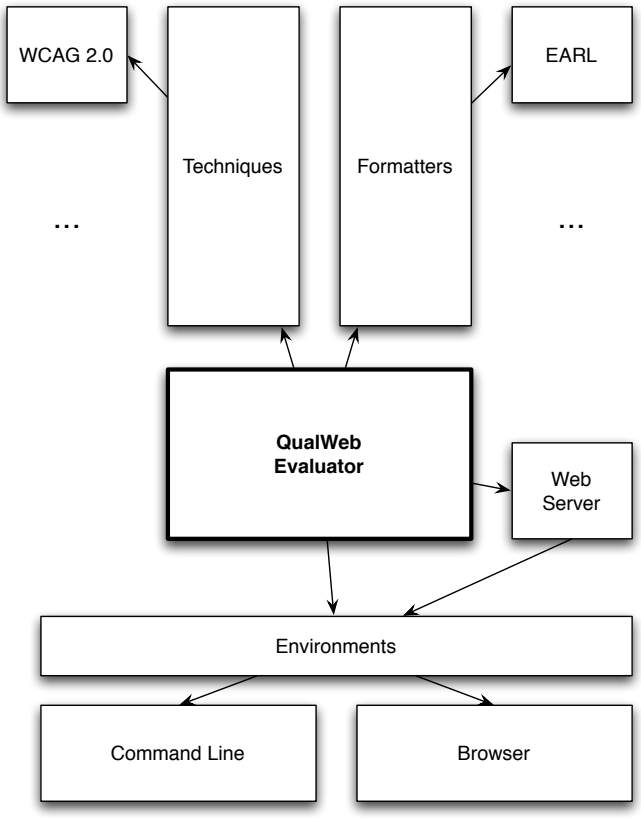

Figure 3: Architecture of the Evaluation Framework

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.

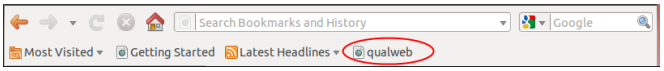

Figure 4: Evaluation execution example on Browser

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.

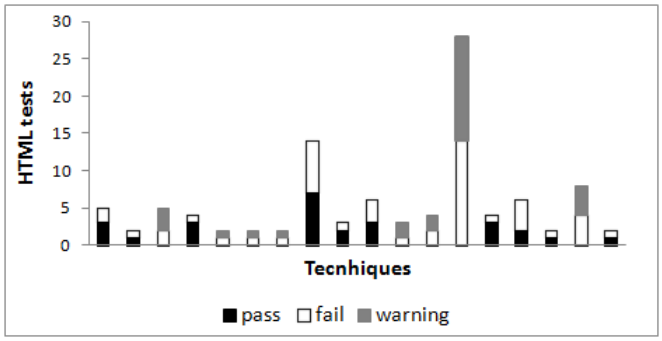

Figure 5: Number of Test Documents per Technique

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

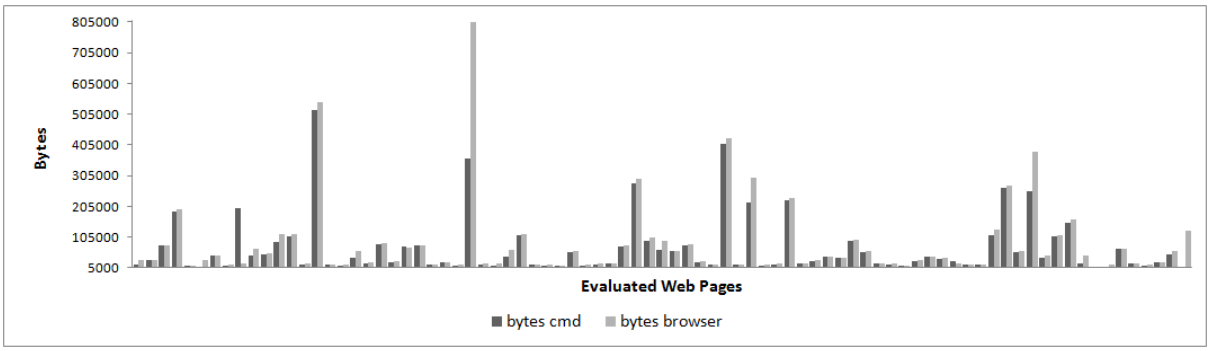

Figure 6: Comparing size in bytes in both environments

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

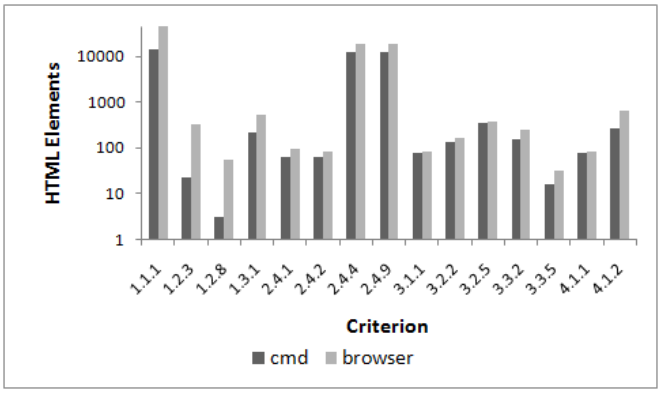

Figure 11: Browser vs Command Line per criterion (log-scale on HTML Elements count)

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

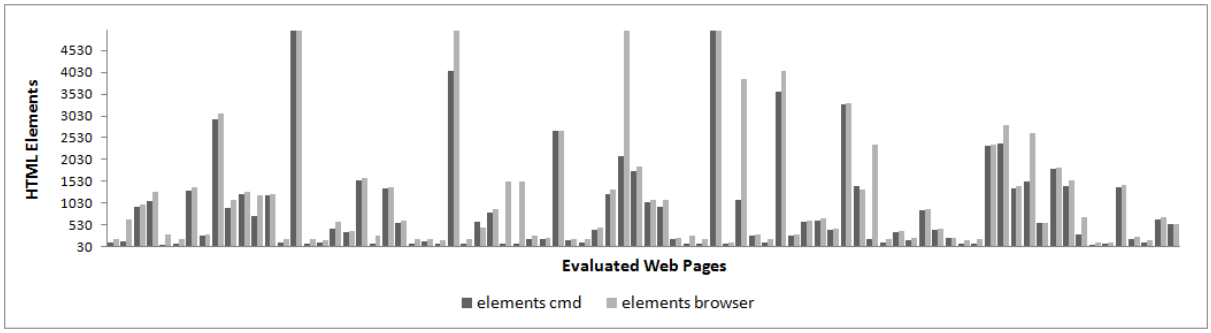

Figure 7: Comparing size in HTML Elements count in both environments

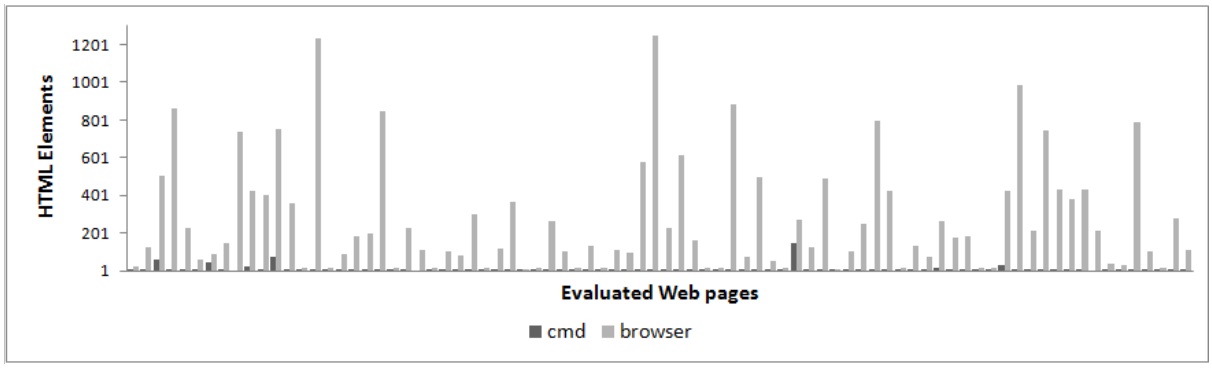

Figure 8: Number of HTML Elements that Passed

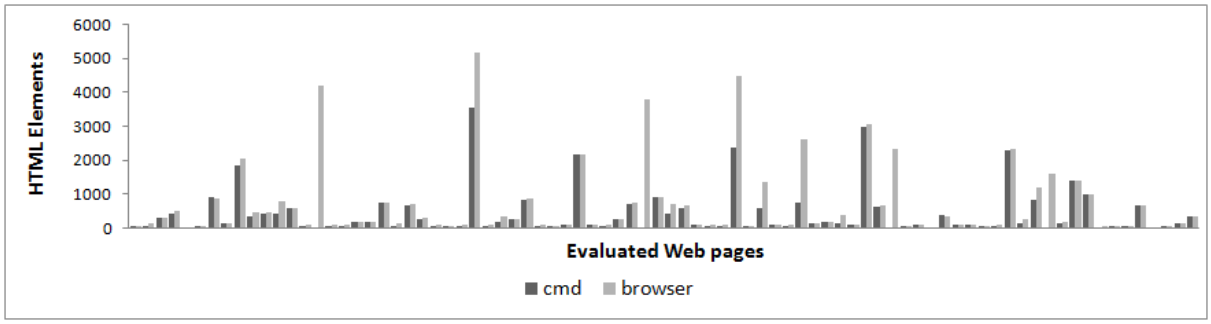

Figure 9: Number of HTML Elements that Failed

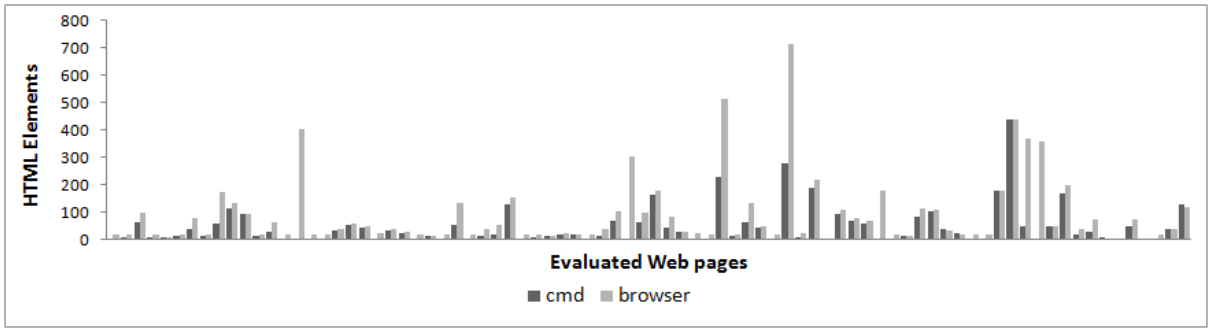

Figure 10: Number of HTML Elements that had Warnings

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;

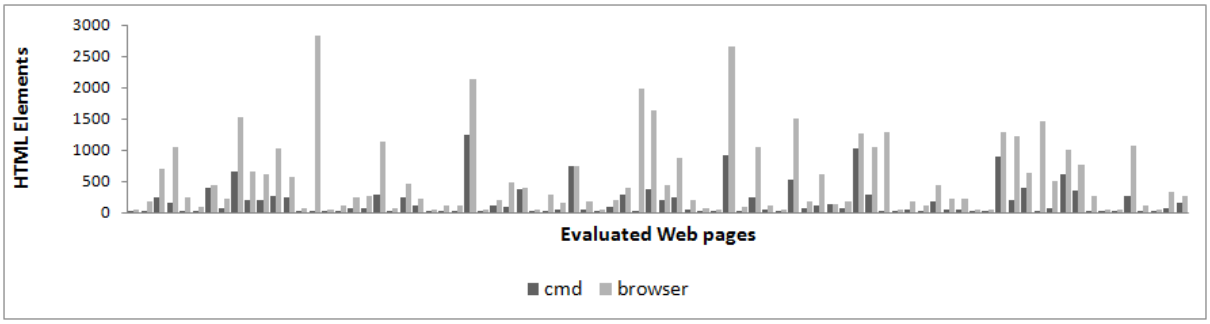

Figure 12: Browser vs Command Line for criterion 1.1.1

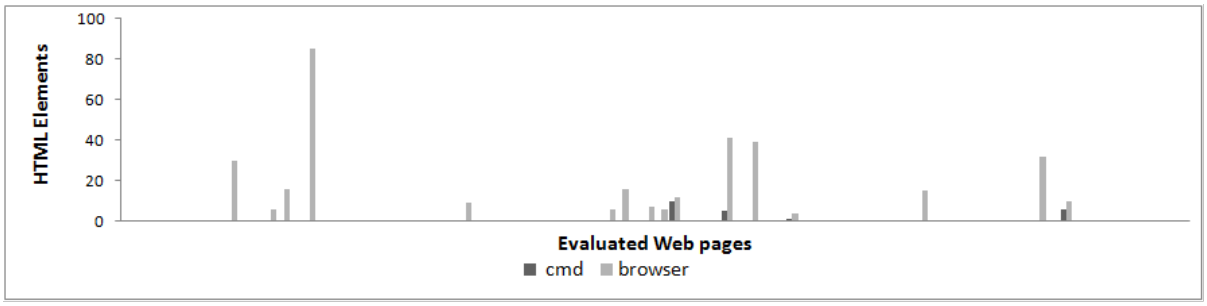

Figure 13: Browser vs Command Line for criterion 1.2.3

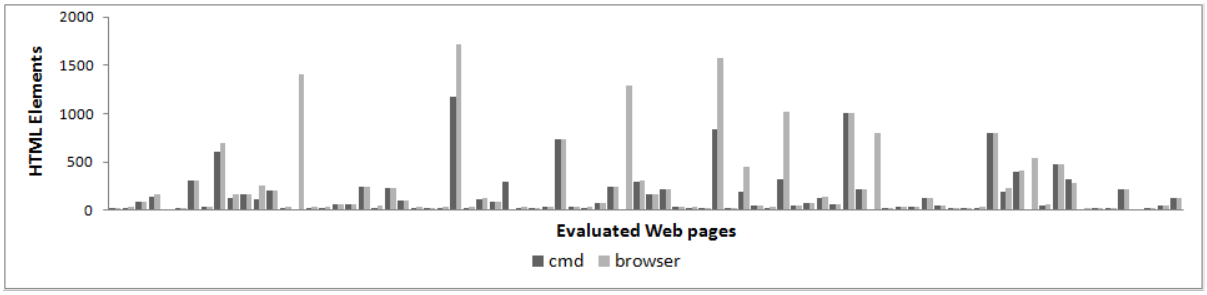

Figure 14: Browser vs Command Line for criterion 2.4.4

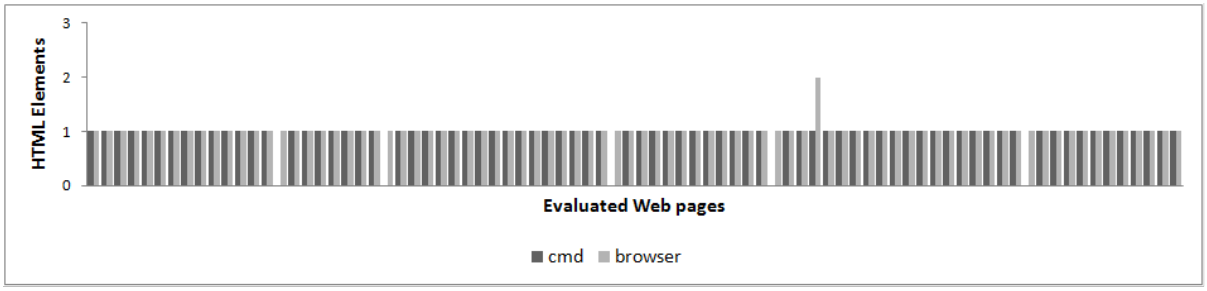

Figure 15: Browser vs Command Line for criterion 3.1.1

- *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.