Journal of Web Engineering, Vol. 7, No.4 (2008) 281-298
© Rinton Press

AN INVESTIGATION OF TOOL SUPPORT FOR ACCESSIBILITY ASSESSMENT THROUGHOUT THE DEVELOPMENT PROCESS OF WEB SITES

JOSEPH XIONG

Institute of Research in Informatics of Toulouse - University Paul Sabatier, Toulouse, France
xiong@irit.fr

MARCO WINCKLER

Institute of Research in Informatics of Toulouse - University Paul Sabatier, Toulouse, France
winckler@irit.fr

Received February 22, 2008
Revised June 23, 2008

This paper investigates the support given by currently available tools for dealing with accessibility at different phases of the development process. At first, we provide a detailed classification of accessibility guidelines according to several levels of automation. Then we analyze which automated inspection techniques are supported by currently available tools for building Web sites. By means of a case study we try to assess the possibility of fixing accessibility problems at early phases of the development process. Our results provide insights for improving current available tools for Web design in order to take accessibility into account at all phases of development process of Web sites.

Key words: accessibility, guidelines inspection, evaluation tools, automated inspection, development process, user interface, Web design, Web sites.

1 Introduction

In recent years Accessibility has become a legal requirement as many countries have enacted laws for Accessibility which are responsible for the content published on the Web [37]. In the United States, the regulation Section 508 [25] was amended in 1998 to address new Information Technologies and the Web. In Europe, the European Commission encourages state members to enact laws for accessibility of public Web sites at all levels of government. Many member states such as France, Germany, Portugal, and the UK, among many others, have created laws for the accessibility of digital content. The European Commission has also supported initiatives for promoting the accessibility such as the creation of a European e-Accessibility Certification (EuraCert) [13] and the development of a Unified Web Evaluation Methodology (UWEM 1.0) [32, 33] whose aims are to provide a set of guidelines and a standard procedure for manual and/or automated accessibility inspections. Whilst UWEM 1.0 and EuraCert represent important steps forward, measurement and certification of Web sites, from a technological point of view are similar to W3C's Web Accessibility Initiatives [38] which already provide guidelines and tools for accessibility checking and digital certificates (e.g. WAI-A, WAI-AA, WAI-AAA) for accessible-compliant Web sites. Accessibility has gained in importance not only because of ethical issues (e.g. e-inclusion) but also because of the opportunities for creating new markets (e.g. elderly customers) and for increasing audience size (e.g. accessible Web not only benefits impaired users but users in general). It therefore became an essential requirement for promoting universal access of Web sites.

Currently, there are many tools for automating guideline inspection of Web sites such as WebXACT [34], TAW [28], WAVE [35], Ocawa [23], and A-Prompt [5]. Most of them include the verification of usability and accessibility guidelines. Even though not all guidelines can be automatically assessed by these tools, they provide valuable help by reducing costs and time of manual inspections. The main inconvenience with these tools is that the evaluation is mostly done on the source code (i.e. HTML/XHTML, CSS) which is only available at latter phases of the development process when the Web site is ready to be deployed.

It is noteworthy that most of ergonomic guidelines do not refer to Web page source code (e.g. HTML elements/attributes or any other particular technology) nor are they limited to the inspection. In fact, according to the phase of the development life cycle different categories of guidelines can be employed for supporting design and/or evaluation of User Interfaces (UIs) over different artifacts [20]. Some ergonomic guidelines such as "*Provide enough colour contrast between foreground and background*" could be applied either on finished Web pages or artifacts such as templates issued from mock-ups (i.e. pencil and paper based prototypes [27]).

The identification and prevention of accessibility problems in early phases might reduce costs of the development. For example, the inspection over a template is usually more efficient and cheaper than the inspections of all pages created from the template itself. Similarly, if inspections could be done on abstract models used to generate the final user interface, then we can ensure accessibility of Web pages by 'construction' and avoid additional work to make pages compliant with accessibility guidelines.

The main goal of this paper is to determine how many accessibility guidelines can be assessed in earlier steps of the development process and how such early assessments might contribute to reduce costs of inspections and improve accessibility of Web sites in general. For this purpose we present hereafter a case study which combines several methods including: i) analysis of tool execution over existing Web sites, ii) evaluation of tool guidance for reporting and/or correcting accessibility flaws whilst editing Web pages, and iii) estimation of checkpoint that could (potentially) be assessed over Web site specifications.

The rest of the paper is organized as follows: Section 2 presents several issues related to the automation of accessibility guidelines, including interpretation of high-level guidelines and levels of automated inspection. Section 3 presents an overview of development tools and evaluation tools currently used to build Web sites. Tools presented in section 3 will be later employed in the case study which is fully described in section 4. Section 5 presents our findings. Section 6 compares our results with related work in literature and finally in section 7 we present conclusions and future work.

2 Accessibility Guidelines

Many guidelines exist [24, 29], most of them include either usability and/or accessibility recommendations. As far accessibility is concerned, the set of accessibility guidelines W3C/WAI WCAG 1.0 [38] is the most widely agreed standard. WCAG 1.0 contains 14 guidelines which express general accessibility principles. WCAG 1.0 guidelines are divided into 65 checkpoints which are reified versions of guidelines describing how they should be interpreted during inspections. Every checkpoint is assigned with a priority level ranging from 1 (high priority) to 3 (low priority). Based on these priorities, a Web application is said to be level "A" conformant if all checkpoints of priority level 1 are satisfied, level "AA" if all checkpoints of priority levels 1 and 2 are satisfied, and level "AAA" conformant if all checkpoints are satisfied. Table 1 illustrates this organization.

Table 1 W3C/WAI WCAG 1.0 organization of guidelines.

Guideline 1: Provide equivalent alternatives to auditory and visual content		
N.	**Checkpoint**	**Priority**
1.1	Provide a text equivalent for every non-text element (e.g., via "alt", "longdesc", or in element content). This includes: images, graphical representations of text (including symbols), image map regions, animations (e.g., animated GIFs), applets and programmatic objects, ascii art, frames, scripts, images used as list bullets, spacers, graphical buttons, sounds (played with or without user interaction), stand-alone audio files, audio tracks of video, and video.	1
1.2	Provide redundant text links for each active region of a server-side image map.	1
1.3	Until user agents can automatically read aloud the text equivalent of a visual track, provide an auditory description of the important information of the visual track of a multimedia presentation.	1
1.4	For any time-based multimedia presentation (e.g., a movie or animation), synchronize equivalent alternatives (e.g., captions or auditory descriptions of the visual track) with the presentation.	1
1.5	Until user agents render text equivalents for client-side image map links, provide redundant text links for each active region of a client-side image map.	3
Guideline 2: Don't rely on color alone		
N.	**Checkpoint**	**Priority**
2.1	Ensure that all information conveyed with color is also available without color, for example from context or markup.	1
2.2	Ensure that foreground and background color combinations provide sufficient contrast when viewed by someone having color deficits or when viewed on a black and white screen.	2

Guidelines are expressed in natural language and quite often they should be interpreted in order to be useful [26]. Due to their nature, guidelines may be more or less suitable for automation. WCAG's checkpoints provide some guidance for employing guidelines in manual inspections. However, automated inspection often requires further interpretation and refinement. For example, let's consider the checkpoint 1.1 "Provide a text equivalent for every non-text element" presented in Table 1. The interpretation on how this checkpoint should be inspected against a Web page may lead to alternative implementations as presented in Table 2. The simplest implementation (i.e. 1.1.1) would just check the presence (or absence) of an alternative text description (i.e. "ALT" attribute). Another possible implementation (i.e. 1.1.2) would consider that “ALT” attributes must not be empty. A third implementation (i.e. 1.1.3) would also verify if text inside “ALT” attributes corresponds to the image they refer to. The best tools implement more than one of these strategies of verification.

Table 2 Example of concrete checkpoints for automated inspections of checkpoint 1.1 of WCAG 1.0.

Checkpoint 1.1 Provide a text equivalent for every non-text element (e.g., via "alt", "longdesc", or in element content)…	
Implementation	**Checkpoint description for automated inspection**
1.1.1	Check the presence of an attribute “ALT” on every tag “IMG”.
1.1.2	Check that attributes “ALT” are not empty.
1.1.3	Check that text inside “ALT” attributes corresponds to the image they refer to.

Currently, there is no available tool supporting the verification of the semantics of alternative text and image’s content, which creates a gap between checkpoint’s descriptions and actual checkpoints automation. However, tools can provide active help pointing out where designers should look (i.e. lines of HTML code) during manual inspections. Characterizing the level of automation helps to estimate the impact of correcting errors when guidelines are violated. Hereafter we present an extended version of Ivory’s taxonomy [19] for describing the levels of automation:

- None: the guideline is supported by the software tool but no automatic analysis is performed (this is typically the case of warnings such as “Use the clearest and simplest language appropriate for a site's content” (source: WCAG 1.0, checkpoint 14.1 [priority 1]);
- Capture: the software tool records data (e.g. collect Web pages weight) that can be used later to evaluate guidelines such as “Web pages should not exceed 70 Kb” (source: AccessiWeb);
- Analysis: the software tool just detects guideline violations but it does not provide any correction for it. For example, the violation of the guideline “Provide a text equivalent for every non-text element” (source: WCAG 1.0, checkpoint 1.1 [priority 1]) would be reported as “img element missing alt attribute, check line 21”;
- Critique: the software tool automatically identifies guideline violations and describes how to proceed to fix them; for example, for guideline: “Clearly identify the primary natural language of a document” (source: WCAG 1.0, Checkpoint 4.3 [Priority 3]) the critique would be: “Check if the content is HTML, check for the html element's lang attribute. If the content is XHTML 1.0, or any version of XHTML served as "text/html", check for both the html element's lang attribute and XML:lang attribute. If the content is XHTML 1.1 or higher and served as type “application/xhtml+xml”, check for the html element’s XML:lang attribute.”;
- Suggestion: the software tool suggests accessibility and/or usability options at design time so that guidelines are respected by tool interposition (or by construction). For example, an accessibility option can be checked to automatically prompt the user to give a textual alternative every time he inserts an image in a page. Another example is the automatic suggestion and consequently creation of accessible tables.

Tools labeled “Critique” evaluate finished artifacts whilst tools labeled “Suggestion” have a proactive role preventing designers to add problems when editing artifacts.

3 Tools for Developing and Evaluating Accessible Web Sites

Tools for Web development are numerous and can be used at different steps of the development process. Development tools are used during the design and implementation of applications while post-implementation tools are used to evaluate or fix problem on the final application. The following categories of tools are considered in this work:

(i) Visual editors and site managers are authoring tools that make Web sites development easier by designing without HTML programming, and also provide features dedicated to site management. Tools in this category typically propose a WYSIWYG editor. Adobe® DreamWeaver®[a] and Microsoft® FrontPage®[b] are such products.

(ii) Model-driven application generators employ several conceptual models to generate the Web applications. Thus, such tools generally cover most of the lifecycle activities and provide a high level of automation. Some examples of tools in this category are: e-Citiz [12], WebRatio [3, 36] and VisualWade [16]. Except e-Citiz, accessibility is not currently supported by these tools.

(iii) Evaluation tools, typically, are used at the end of the development process to assess the final Web application in order to detect usability and/or accessibility problems. There are several categories of evaluation tools [19]: log analysis tools, automated inspection tools, HTML syntax validators, etc. If all these categories may support accessibility evaluation or some aspects of accessibility evaluation, guideline inspection tools are certainly the most powerful and the most widespread for assessing Web accessibility. Among the many products in this category are WebXACT [34], TAW [28] and Ocawa [23].

One may notice that we did not consider tools conceived to simplify the integration of database queries within a Web page (e.g. Web-DBPL integrators [15]) because such tools do not offer more accessibility features than visual editors and site managers. One might consider the use of the underlying relational model of the database applications to drive the user interface definition including description of functionalities and navigation. However, the use of metadata as input for the user interface definition requires a specific work on both transformation methods which is out the scope of this paper.

4 Parameters of the Case Study

For the purposes of this study, only the WCAG 1.0 guidelines were considered. Moreover, the analysis was limited to the level AA guidelines (i.e. checkpoints covering priorities 1 and 2) making a total of 49 individual checkpoints. Nine (9) tools covering different phases of the development were selected from a large set of tools available for the categories described in section 3. Due to the nature of the

[a] Available at: http://www.adobe.com/products/dreamweaver/

[b] Available at: http://www.microsoft.com/frontpage/. Frontage has been recently replaced by two new tools (Office SharePoint® Designer 2007 and Expression™ Web Designer) but this updates has no impact on the discussion carried out on this paper.

tools analyzed, different methods of assessment were employed. The analysis of checkpoints, inspection of tools and the artifacts produced by tools allowed the estimation of the number of checkpoints covered by tools, number of checkpoints that could be evaluated at each phase of the development process, estimation of the potential effort for correcting accessibility problems in early phases of the development process.

4.1 Tools analyzed

Yet simplified, a development process is required to understand when tools are used. Although there is no consensus on phases of development process, one might agree on generic phases such as specification (abstract modelling of application), design (models refinement including design constraints, integration of content, visual design of elements, etc.), implementation (production of application code, database creation, etc.), post-implementation (testing and maintenance). As depicted in Figure 1, some tools can be used in a specific phase of the development process (e.g. WebXACT, TAW and Ocawa, at post-implementation phase), whilst other span several phases (e.g. Frontpage, Dreamweaver, VisualWade and SketchiXML) or provide some support during the entire development process (e.g. WebRatio and e-Citiz).

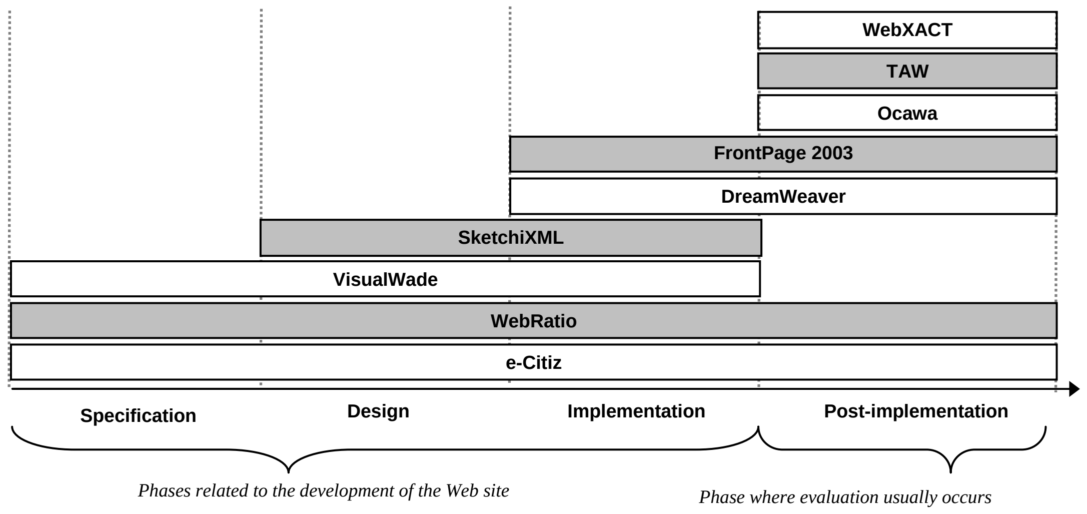


Figure 1. Tools support according to steps in development process.

The notion of activities performed and tool support at each phase is important to understand where guidelines should be taken into account in the development process. Tools, like VisualWade, which generates HTML code from specification, should implement accessibility policies very early in the development process to ensure the generation of accessibility problems on generated HTML code. These tools address all the categories presented in section 3: Visual editors and site managers (i.e. Frontpage, Dreamweaver), Model-driven application generators (VisualWade, SketchiXML, WebRatio and e-Citiz) and Evaluation tools (i.e. WebXACT, TAW and Ocawa).

e-Citiz [12], WebRatio [3, 36], and VisualWade [16] follow a model-driven approach which make them suitable in early phases of development process. WebRatio and VisualWade are generic tools for specification of Web sites whilst e-Citiz is a framework dedicated to the specification and deployment of electronic administrative procedure (or e-procurement). One of the reasons that e-Citiz was included in this survey is the fact that it implements accessibility policies to generate e-procedures compliant with WCAG 1.0 guidelines.

SketchiXML [11] is a representative tool for the activities carried out in the design phase. This tool allows a designer to rapidly draw a user interface such as with a classic graphics painting program. SketchiXML is equipped with a text and shape recognition engine allowing automatic identification of sketches and the automatic reconstruction of user interface components accordingly. The interface is specified using the UsiXML notation [31]. Thus, a mock-up reflecting the final interface of a Web page is easily obtained with SketchiXML. Indeed, even if SketchiXML has no features dedicated to accessibility conformity, evaluations could be performed on the UsiXML source code using a special module called UsabilityAdviser[c]. However, the results presented in this study do not take into account the UsabilityAdviser plug-in.

Macromedia DreamWeaver [4] and Microsoft FrontPage 2003 [22] are among the most popular tools for developing Web sites. These tools provide many facilities such as a WYSIWYG programming style, CSS styles editing, semi-automatic generation of code for complex structure (tables, forms, etc.), pre-designed block of scripts code (Javascript, Flash, etc.) that help to ensure a high level of productivity even for novice Web designers. Dreamweaver and FrontPage both include accessibility features that will help conforming to accessibility guidelines while developing the application. These tools can be coupled the LIFT Machine[d] for inspecting the usability and accessibility. The combined use of DreamWeaver/ FrontPage with the LIFT Machine would create a bias in our study because it would give a false positive score for tools whose main goal is to support the edition of web pages. For this reason, the LIFT plug-in was not included in this study.

WebXACT [34], TAW [28] and Ocawa [23] are well known tools for accessibility evaluation. The purpose of these evaluation tools is to evaluate the accessibility of any Web application. All these tools are online Web services that parse the HTML source code of pages and produce an evaluation report where accessibility errors and warnings are mentioned.

4.2 Methods employed

Due to the different purposes and use of the tools, three evaluation methods were applied: i) analysis of tool execution over real Web sites, ii) evaluation of tool guidance for reporting and/or correcting accessibility flaws whilst editing Web pages and iii) estimation of the potential for automated checkpoints inspection over Web site specifications. Figure 2 shows the decision processes for selecting the best evaluation method in which case.

The analysis of tool execution was performed over 12 real Web sites known by their accessibility defects. The level of automation of inspection was measured against the reports provided by tools. This

[c] Available at: http://www.usixml.org/index.php?mod=pages&id=42 (June 2008)
[d] Available at: http://www.usablenet.com/usablenet_liftmachine.html (June 2008)

method was used with evaluation tools (i.e. TAW, Ocawa and WebXACT) and it reflects the results that can be obtained by the current practice.

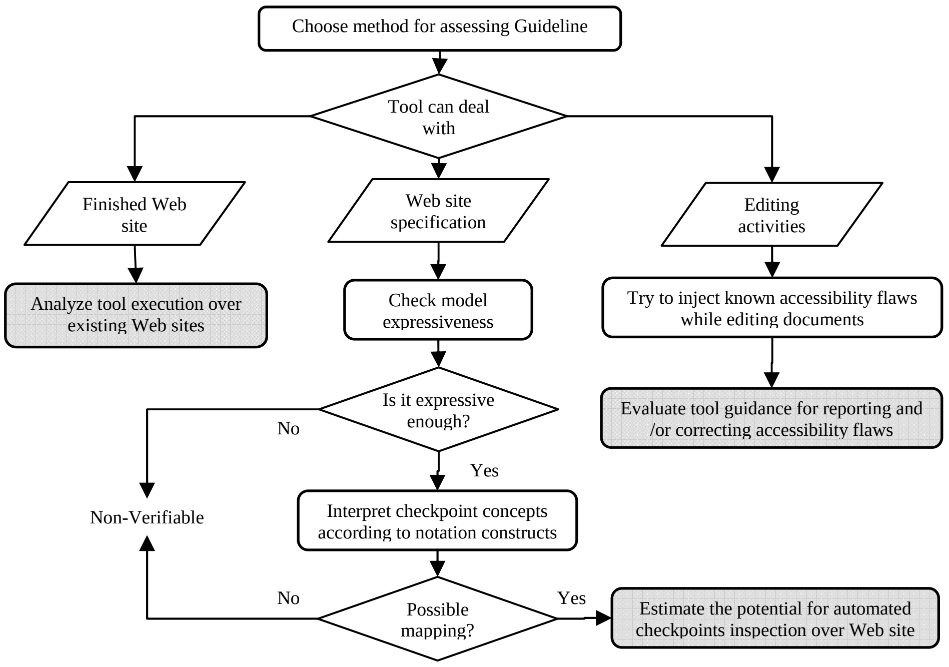

Choose method for assessing Guideline
Tool can deal with
Finished Web site
Web site specification
Editing activities
Analyze tool execution over existing Web sites
Check model expressiveness
Try to inject known accessibility flaws while editing documents
Is it expressive enough?
No
Yes
Evaluate tool guidance for reporting and /or correcting accessibility flaws
Non-Verifiable
Interpret checkpoint concepts according to notation constructs
No
Possible mapping?
Yes
Estimate the potential for automated checkpoints inspection over Web site

Figure 2. Decision process applied to select methods for assessing tool support.

The second method evaluates the tool guidance for reporting and/or correcting accessibility flaws whilst editing Web pages. Several techniques such as active help and automatic correction are analyzed with respect to treatment given to accessibility checkpoints. It was used with FrontPage 2003 and Dreamweaver. Such an analysis was based on the availability of components (e.g. images, page, etc.).

The estimation of checkpoints inspections was used in tools supporting a model-based approach (i.e. SketchiXML, VisualWade, WebRatio and e-Citiz) when only artifacts were available (e.g. conceptual models). All artifacts produced with these tools were analyzed in order to determine if the information they contain can be used (or not) for supporting inspection of accessibility guidelines. For example, object oriented classes describing pages could be interpreted as Web pages, arrows as links and so on. If the specification supports such semantics, we interpret and apply the appropriate checkpoints to inspect classes and arrows as if they were pages and links.

5 Results of Tools Assessment

This section summarizes our results. Table 3 presents the full description of our findings. In the subsequent sections a synthesis and critical analysis of such data is provided.

Table 3 Number of WCAG 1.0 checkpoints supported.

		Model-driven application generators				Visual editors/site		Evaluation tools		
Priority	**N. checkpoint**	**e-Citiz**	**WebRatio**	**VisualWade**	**SketchiXML**	**DreamWeaver**	**FrontPage 2003**	**Ocawa**	**TAW**	**WebXACT**
1	1.1	An		Su	Su	Su	No	An	Cr	Cr
	1.2							An	An	An
	1.3									
	1.4									An
	2.1								An	An
	4.1								No	No
	5.1	Su			Su		No	An	An	An
	5.2				Su				An	An
	6.1								No	No
	6.2								An	
	6.3								An	An
	7.1								No	No
	8.1								An	An
	9.1								An	An
	11.4								No	No
	12.1	An	Su	Su	Su	Su	No	An	An	An
	12.4	Su		Su		Su				An
	14.1								No	No
2	2.2	An	No		An				An	An
	3.1	An							No	No
	3.2					No	Su		No	No
	3.3	Su	Su	Su					Cr	No
	3.4	An	An						An	An
	3.5								No	No
	3.6								No	
	3.7								No	No
	5.3								Cr	Cr
	5.4								An	An
	5.5	An	An		Su	Su		An	An	An
	6.4								An	No
	6.5									
	7.2								No	Cr
	7.3								No	An
	7.4							An	An	An
	7.5							An	An	An
	8.1								Cr	
	9.2								An	An
	9.3							An	An	An
	10.1							An	Cr	Cr
	10.2			Su	No	Su			No	No
	11.1								No	No
	11.2							An	Cr	Cr
	12.2		Su	Su						No
	12.3	Su	No		No				No	No
	12.4	Su		Su		Su	No	An	Cr	An
	13.1	An			No	No	No	No	No	No
	13.2	Su	Su	Su					No	
	13.3		Su	Su					No	No
	13.4	Su	Su	Su					No	No
Number of Checkpoints		14	10	10	9	8	6	12	44	43
% checkpoints supported		28.57%	20.41%	20.41%	18.37%	16.33%	12.24%	24.49%	89.8%	87.76%

Legend

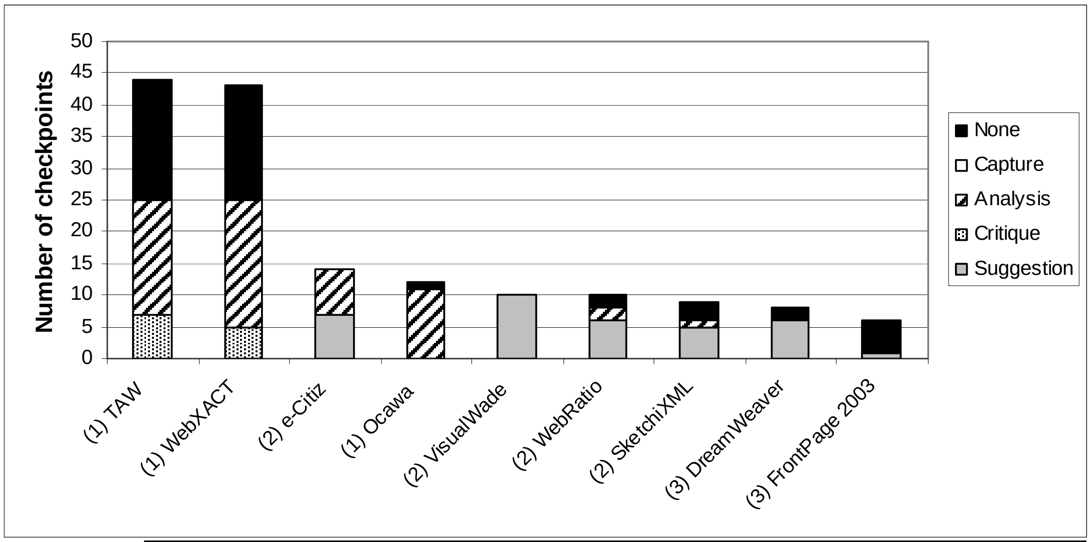
No	None
Ca	Capture
An	Analysis
Cr	Critique
Su	Suggestion

☐	Non-relevant for the domain
░	Non-verifiable
▨	Non-supported
▒	Poorly supported
▩	Correctly supported
■	Fully supported

5.1 Number of WCAG 1.0 checkpoints supported

The number of checkpoints actually supported by tools devoted to evaluation is higher than those supported by development tools, as expected. As depicted by Figure 3, TAW presents the best performance, followed by WebXACT, covering 44 of the 49 WCAG 1.0 checkpoints (i.e. priority 1 and 2 checkpoints). We recall that results were obtained by different methods. In Figure 3, the numbers in parenthesis indicate the tool category and the method of analysis employed (i.e. (1) execution of evaluation tools, (2) estimation based on artifacts produced by model-driven tools, (3) analysis provided during edition of pages). The tools are ordered by the number of checkpoints supported.

It can be seen that Ocawa is only able to check (12) of the 49 checkpoints, which means the worst performance among evaluation tools (i.e. category 1). This is explained by the fact Ocawa only considers checkpoints which can be automatically detected (i.e. automation level analysis). However, when comparing Ocawa performance to TAW and WebXACT, there is a difference in number of checkpoints inspected for the automation level analysis. None of the tools assessed supports analysis based on data capture.

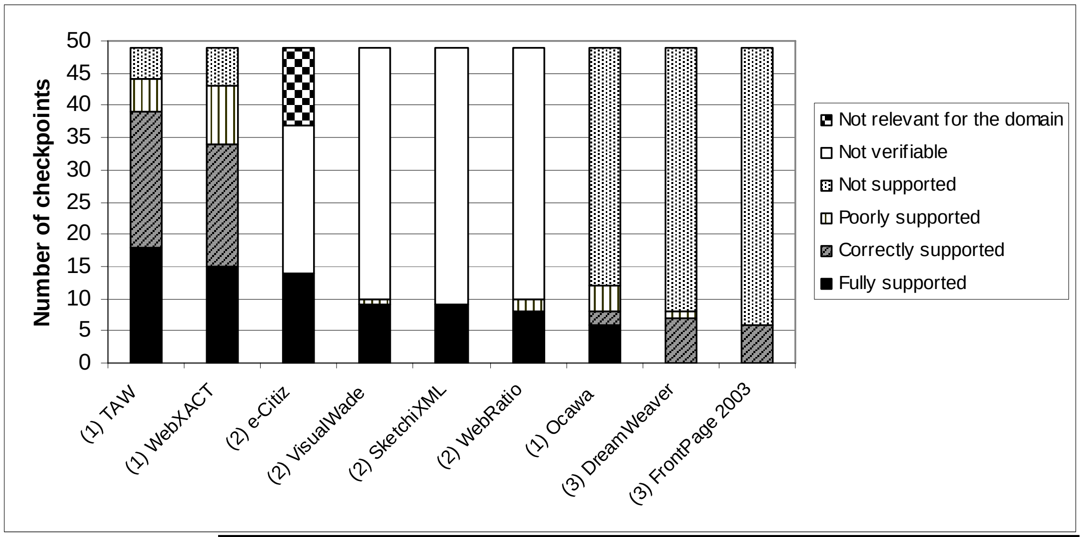


	TAW	WebXACT	e-Citiz	Ocawa	VisualWade	WebRatio	SketchiXML	DreamWeaver	FrontPage 2003
Suggestion	0	0	7	0	10	6	5	6	1
Critique	7	5	0	0	0	0	0	0	0
Analysis	18	20	7	11	0	2	1	0	0
Capture	0	0	0	0	0	0	0	0	0
None	19	18	0	1	0	2	3	2	5
Total	**44**	**43**	**14**	**12**	**10**	**10**	**9**	**8**	**6**

Figure 3. Number of WCAG 1.0 checkpoints according the automation Level.

5.2 Analyzing quality of checkpoint support

The support provided by tools can be different according to the information available in artifacts. The checkpoint “Provide alternative text for images” can be fully supported over HTML but it becomes non-verifiable over navigation models, for instance. In addition, some tools automatically detect errors (i.e. guideline violation) whilst others just raise warnings informing designers to proceed to manual inspection. In order to measure the quality of inspections we establish some criteria based on empirical evidence using the tools. Thus, Figure 4 provides a distribution of checkpoints according to such a qualitative evaluation. From left to right, we present the tools according to the quality of support.



	TAW	WebXACT	e-Citiz	Ocawa	VisualWade	WebRatio	SketchiXML	DreamWeaver	FrontPage 2003
Fully supported	18	15	14	9	9	8	6	0	0
Correctly supported	21	19	0	0	0	0	2	7	6
Poorly supported	5	9	0	1	0	2	4	1	0
Not supported	5	6	0	0	0	0	37	41	43
Not verifiable	0	0	23	39	40	39	0	0	0
Not relevant	0	0	12	0	0	0	0	0	0
Total	**44**	**43**	**14**	**12**	**10**	**10**	**9**	**8**	**6**

Figure 4. Distribution of checkpoints according to qualitative support.

Hereafter we provide a detailed description of criteria:

- Fully supported: all facets of the checkpoint are taken into account. Not only is an inspection done but extra features such as examples and/or best practices are provided.

- Correctly supported: the checkpoint is supported by the tool but does not tackle all the evaluations that are possible. Example: only highlighting errors/warnings while automatic correction could be done.
- Poorly supported: only errors are reported, not warnings; no explanations are given; no additional documentation for corrections is given; etc.
- Non-supported: the checkpoint can be verified but is not implemented by the tool.
- Non-verifiable: the checkpoint cannot be verified even by a human expert, because it requires information that is missing. An example of such checkpoints is 14.1: “Use the clearest and simplest language appropriate for a site’s content”. At the specification phase this checkpoint is not verifiable because texts contained in Web pages are not yet available.
- Non-relevant for the domain: checkpoints are not taken into account, deliberately, because elements never appear in the application domain addressed by the tool.

5.3 Assessing guidelines support throughout the development process

In order to provide a rough estimation of the potential for accessibility evaluation of tools in this survey Figure 5 shows the best tool’s score at each phase according to the number of checkpoints supported (see Figure 3). So that, we have 14 checkpoints supported by e-Citiz at specification phase, 9 checkpoints supported by SketchiXML at the design phase, 10 checkpoints supported by both VisualWade and WebRatio at the implementation phase and 44 checkpoints supported by TAW at the post-implementation phase. Such an estimate should be taken carefully because it was made considering different methods. Moreover, these scores do not reflect any real development process combining the synergistic use of these tools. However, it provides some insights about the potential of currently available artifacts to support automated inspection.

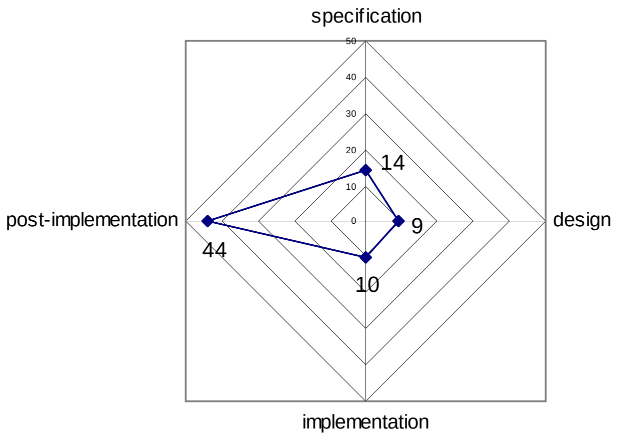


Figure 5. Number of checkpoints supported during the development process.

It is important to say that this study is based on existing tool support and the evaluation reflects the actual use of such tools. For example, at the phase of specification 28.7% (i.e. 14) of WCAG 1.0 checkpoints are fully supported by the tool e-Citiz. From the specification e-Citiz automatically generates the final user interface (thus skipping most activities related to design and implementation) and it does not support any other particular design activity. This lower number at the design phase is due to the kind of artifacts used in the analysis. One must consider that according to the artifacts manipulated the results would change. The proactive help to avoid the introduction of accessibility flaws whilst editing Web page was estimated to 18.37% (i.e. 9) of the checkpoints. This low score at the design phase does not mean that evaluations at this stage of the development process are less important. It reflects the poor support of existing tools for design activities.

5.4 Estimating correction effort

This section presents a discussion on the correction effort of checkpoints violation. The calculation must consider the impact of the correction on the maintenance of the Web sites which might require combined manual actions and automated actions. So generally speaking, the effort required to perform these actions depends on the tool support available. It is possible to distinguish the following dimensions affecting the final correction costs:

- Actions that should be performed to ensure the conformity with that checkpoint; for example, *add information* (e.g. add an alternative text for an image), *delete information* (e.g. remove the line code for automatic page refresh as this may disorient users) and *modify existing information* (e.g. in a form, labels and their controls must be visually associated; to ensure the conformity with that checkpoint, developers might have to modify the position of labels).
- Automation level available to fix the checkpoints violations which may include *none* (i.e. manual correction), *interactive* (tools can *capture* useful data for correcting the problem and/or support *analysis* and *critique* of the UI pointing where problems occurs but the correction is done manually), *automated correction*, and *embedded* in the tool (e.g. *proactive* help) so that it prevents designers to add flaws (e.g. prevent to saving pages without a web page title).
- Integration level between errors reported and evaluated application. For example: *no integration* (reports are presented apart from the application), *linked reports* (e.g. it is possible to navigate from reports to the specific position in the code where errors occur), *continuous report* (e.g. reports are updated whenever the Web page is modified).
- Knock-on effect of fixing checkpoints violations. The correction of a problem in early phases of the development process will prevent it to reappear in the future. For example, fixing color contrast on templates will prevent contrast corrections in all Web pages. This dimension is directly related to the step of the development process when the problem is identified and corrected.
- Frequency of checkpoints violations. The global effort should take into account the frequency of problems; for example, a missing page title might only appear once for every Web page whilst missing alternative text for images will appear as many times as images in the Web site.

It seems clear that early evaluations would reduce the cost of correction effort associated to negative knock-on effect of checkpoint violations. A practical measurement that should be taken into account is the number of occurrences of errors and warnings that are identified when a checkpoint is not respected. To exemplify this situation, Table 4 shows the results of an automated evaluation using WebXACT of two Web sites (A and B). One can remark that WebXACT was able to detect 27 violations (7 errors + 20 warnings) for the site A and 24 violations (4 errors + 20 warnings) for the site

Web B. This evaluation demonstrates the negative knock-on effect of violations corresponding to ~339 occurrences on the site A and to 385 on the site B.

Table 4 Average number of checkpoints violation (errors + warnings) for two web sites.

Web site	Errors	Warnings	Violations	Occurrences of Errors	Occurrences of Warnings	Occurrences of violations
A	7	20	27	142	197	339
B	4	20	24	197	188	385

Not all checkpoints can be inspected in early phases. Moreover, the actions required to fix a checkpoint violation may change according to the phase of the development process, thus affecting the estimation effort. To exemplify this, let’s consider the effect of early corrections on the following checkpoints:

- Checkpoint 5.1 (i.e. “Every table should have a proper header for its cells”). This checkpoint is automatically taken into account at specification phase by exploiting abstract models which can represent collections of data suitable to be implemented as tables. Artifacts produced at design phase do not provide any further information for inspection of headers. At implementation phase, this checkpoint can be automatically checked by editor tools when creating new tables. At evaluation phase, this checkpoint can be automatically detected by analyzing the HTML code and the correction would require the creation of headers (action add) and introduction of appropriate labels (action add) but it should be manually fixed.
- Checkpoint 12.4 (i.e. “Create a direct association among labels and form fields”). This checkpoint can be automatically taken into account at specification and implementation phases. At the design phase, designers must correct manually the positioning of elements; due to the fact that creative aspect of design is not suitable to do these corrections automatically. At post-implementation this problem can be automatically detected but if the checkpoint is violated developers should create a label (action add id to the label) and establish an association between internal label and the field (action add cross-reference to the field).
- Checkpoint 13.4 (i.e. “Provide coherent visual mechanism across pages”). By the means of appropriate graphical templates, this correction can be automatically made at implementation and specification phase. The correction at design and post-implementation phases will require manual intervention to correctly place navigation elements in pages (action modify position).

Assuming that the cost of individual actions is always the same, it is possible to determine that the effort required to correct problems related to the checkpoints 5.1 and 12.4 is higher in post-implementation phases than in earlier phases. For the checkpoint 13.4 the effort is the same at design and post-implementation phase. The analysis presented above is far from presenting a definite measurement method of costs associated to the correction of violations of accessibility checkpoints. Moreover, it is very simplistic as it does not take into account the cost of developing the evaluation tools. However, they illustrate that some checkpoints (e.g. WCAG 1.0 5.1, 12.4 and 13.4) can effectively be inspected in early phases of the development process thus reducing the impact of the multiplicative effect of correction of final applications. Additionally, we suggest that early evaluation and in particular proactive help would have a positive impact for training designers to implement best practices concerning accessibility and usability of user interfaces.

6 Discussion and Related Work

Using accessibility guidelines is not straightforward for developers and evaluators because some expertise is required to correctly understand and apply the guidelines. In addition, manual inspections are tedious and costly. To overcome these limitations several tools for organizing guidelines, guiding the manual inspection and supporting automated inspection have been developed [29]. Currently, several tools exist to assist evaluators during the inspections of Web applications by automatically detecting problems, flexibly reporting errors and in some case repairing accessibility flaws. TAW [28] and WebXACT [34] provide online inspection of Web sites and became one of the most popular tools currently employed. In addition to automated inspection of guidelines, A-Prompt [5] automatically repairs some accessibility flaws. Tools like EvalIris and its successor EvalAccess [1,2], MAGENTA [21] and DESTINE [6] also provide flexible guidelines selection thus tailoring the inspection to a specific sub-set of guidelines. Many of these tools have evolved in more recent years towards tool independence of guideline descriptions by the means of external languages (most of them based on XML schema). However, there is not yet a general consensus on the format of guideline descriptions. For example, WebAccessibility Evaluator (WAEX) [9] supports the automated inspection of guidelines expressed in XSLT grammars; KWARESMI [30] and its successor DESTINE [6] can inspect guidelines described by the Guideline Description Language (GDL) and MAGENTA [21] inspect guidelines described by the Guideline Abstraction language (GAL).

One advantage of automated inspection is that no extensive knowledge on ergonomics is required since most of the technical details are embedded into the tools. However, due to the inner nature of guidelines, it seems clear that not all checkpoints can be fully automated and some of them still require interpretation and manual inspection. For graphical user interfaces, the automation limit has been estimated to 44% of guidelines [14]. For Web applications, the automation limit of full inspection has been estimated by Cooper at al. to 15% with some partial support to 35% of guidelines [10]. The results presented in the paper study are very similar, indeed. When we look at Table 3 the *full analysis support* (i.e. combination of full-support + analysis, legend **An**) provided by evaluation tools, we can found an automation limit ranging from 16.33% for TAW (i.e. 8 checkpoints) to 18.36% (i.e. 9 checkpoints) which is close to the *15% full support* reported by Cooper at al [10]. Similarly, when counting the number of checkpoints non-supported (legend No) and those that due to the inner nature are not supported (legend ▨), we can estimate the manual support to 49% for both TAW and WebXACT.

Traditionally, evaluations are performed over advanced prototypes. However, it has often been argued that early evaluations can reduce the costs of correction in latter phases by preventing negative knock-on effects of errors in specifications [6, 17, 39]. Currently, some tools for automated inspection of Web applications exist but most of them only provide support at post-development phases of the development process. Some previous works [8, 19] have compared the efficiency of tools for supporting accessibility evaluation. However, these studies were limited to the evaluation of finished Web sites. There is an increasing interest on early inspections of usability and accessibility of the user interface and new evaluation methods have been proposed to face with the inner difficulties of predicting potential usability/accessibility flaws. For this purpose, new methods based on the inspection of prototypes [297] or model-based specifications [7, 39] have been proposed. The quality of evaluations in early phases of the development depends upon the quality of specifications available (e.g. prototyping, models, etc). However, the maturation of model-based approaches for user interface design created new opportunities for automated evaluation. We found that most checkpoints could be inspected in early phases and could also be integrated into development tools, thus preventing designers to introduce by mistake accessibility problems into design. A typical example of this is the checkpoint "creation of associations between labels in fields" which can be either automatically inspected but also be used as start point to generate accessible final code of Web sites. We also suggest that this proactive strategy applied in early phases of development would help designers to think about accessibility which would have a positive effect on the general quality of the user interface.

Currently there are several guidelines sources for user interfaces [18, 25, 26]. However, as far as technical accessibility is concerned, the WCAG 1.0 [38] is still the most widely agreed source of Web accessibility guidelines on which actual European and many national certifications of Web sites are based on [13]. Automated inspections based of WCAG 1.0 guidelines only provide a basic support for ensuring technical accessibility which does not prevent the occurrence of problems during the use of the application by people with disabilities (i.e. real accessibility). So that automated inspections must be combined with other evaluation techniques (e.g. user testing) for a better coverage of accessibility problems. Notwithstanding, this case study shows that an important number of guidelines could be taken into account in early phases of the development and in particular during phases of the development process where the participation of users is not straightforward (e.g. specification phases).

7 Conclusions

This work assesses current available support for anticipating inspections of Web applications with respect to accessibility guidelines by combining analysis of existing evaluation tools and estimating the possibility of inspection over artifacts produced by tools. Concerning the analysis over artifacts, it is important to say that results will certainly change according to the kind of information artifacts contain. A feasibility study could determine which information could be added into artifacts (i.e. abstract models) to increase the number of accessibility checkpoints inspected. We assume that ensuring accessibility conformity should be a concern at each phase of the development process and consequently it should be integrated as soon as possible in the lifecycle. The analysis performed shows that it is possible to evaluate some guidelines in very early phases of the development process. Even if the most fully automated of all guideline inspection techniques is not possible in early phases some proactive help can help designers to prevent errors. The results presented in this paper are part of a preliminary study on cost/benefits of anticipating inspections of accessibility guidelines in early phases of the development process. Currently it is very difficult to estimate the correction effort of accessibility problems because there is a lack of metrics required to measure activities related to fixing bugs on the user interface. Some problems could be easily fixed by changing an attribute of a tag (ex. by adding an alternative text to an image) while others would require an entire reengineering of Web page (ex. remove tables used to do the page layout). The correction effort should also take into account the multiplicative effect of errors (that could be detected in early phases, for example on a template) that are repeated in all pages of the Web site.

Acknowledgements

This research is supported by COST294-MAUSE (http://cost294.org/), ANRT-CIFRE and Genigraph (http://www.genigraph.fr), Genitech Group.

References

1. Abascal, J., Arrue, M., Fajardo, I., Garay, N., and Tomás, J. 2004. The use of guidelines to automatically verify Web accessibility. Univers. Access Inf. Soc. 3, 1 (Mar. 2004), 71-79.
2. Abascal J.; Arrue M.; Fajardo I.; Garay N. An expert-based usability evaluation of the EvalAccess web service. In R. Navarro-Prieto, J. Lorés (Eds.): HCI related papers of Interacción 2004, Springer, Netherlands, 2006, pp. 1-17.
3. Acerbis, R., Bongio, A., Butti, S., Ceri, S., Ciapessoni, F., Conserva, C., Fraternali, P., Carughi, G. T. WebRatio, an Innovative Technology for Web Application Development. In proceedings of ICWE’2004, Munich, Germany; July 28-30th 2004.
4. Adobe Dreamweaver CS3. http://www.adobe.com/fr/products/dreamweaver/
5. A-Prompt, Web Accessibility Verifier, http://aprompt.snow.utoronto.ca/
6. Beirekdar, A., Keita, M., Noirhomme-Fraiture, M., Randolet,F. Vanderdonckt, J., Mariage, C. DESTINE: Flexible Reporting for Automated Usability and Accessibility Evaluation of Web Sites. INTERACT 2005: 281-294.
7. Bernhaupt, R., Navarre, D., Palanque, P., Winckler, M. Model-Based Evaluation: A New Way to Support Usability Evaluation of Multimodal Interactive Applications. In Maturing Usability: Quality in Software, Interaction and Quality (chapter 5). Springer Verlag (Human-Computer Interaction Series), October 2007, Law E., Thora Hvannberg E., Cockton G. & Vanderdonckt J. (Eds.). pages 96-122.
8. Brajnik, G. 2004. Comparing accessibility evaluation tools: a method for tool effectiveness. Univers. Access Inf. Soc. 3, 3 (Oct. 2004), 252-263.
9. Centeno, V.; Kloos, C.; del Toro, J. M. B.; Gaedke, M.Web Accessibility Evaluation Via XSLT. WISE Workshops 2007: 459-469.
10. Cooper, M., Limbourg, Q., Mariage, C., Vanderdonckt, J., Integrating Universal Design into a Global Approach for Managing Very Large Web Sites, Proc. of the 5th ERCIM Workshop on User Interfaces for All UI4ALL’99 (Dagstuhl, 28 November-1 December 1999), A. Kobsa & C. Stephanidis (eds.), GMD Report 74, GMD - Forschungszentrum Informationstechnik GmbH, Sankt Augustin, 1999, pp. 131-150. Available at: http://ui4all.ics.forth.gr/UI4ALL-99/Cooper.pdf
11. Coyette, A., Kieffer, S., Vanderdonckt, J., Multi-fidelity Prototyping of User Interfaces, in Proc. of 11th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT’2007 (Rio de Janeiro, 10-14 September 2007), Springer-Verlag, LNCS Vol. 4662, 2007, pp. 149-162.
12. e-Citiz, Le monde des téléservices à portée de clic. http://www.e-citiz.com/
13. European eAccessibility Certification. Available at: http://www.euracert.org/
14. Farenc, Ch., Liberati, V., Barthet, M.-F., Automatic Ergonomic Evaluation: What are the Limits?, Proc. of 2nd Int. Workshop on Computer-Aided Design of User Interfaces CADUI’96 (Namur, 5-7 June 1996), J. Vanderdonckt (ed.), Presses Universitaires de Namur, Namur, 1996.
15. Fraternali, P.: Tools and approaches for developing data-intensive Web applications: A survey. ACM Computing Surveys, 31, 227-263, (1999).
16. Gomez, J. Model-Driven Web Development with VisualWADE. In proceedings of ICWE’2004, Munich, Germany; July 28-30, 2004. pp. 611-612.
17. Hornbæk, K., Høegh, R. T., Pedersen, M. B., Stage, J. Use Case Evaluation (UCE): A Method for Early Usability Evaluation in Software Development. in Proc. of 11th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT’2007 (Rio de Janeiro, 10-14 September 2007), Springer-Verlag, LNCS Vol. 4662, 2007, pp. 578-591.

18. ISO IS 9241-171 Ergonomics of Human-System Interaction: Guidance on software accessibility.
19. Ivory, M.Y., Automated Web Site Evaluation: Researchers' and Practitioners' Perspectives, Kluwer Academic Publishers, Dordrecht, 2003.
20. Javaux, D., Colard, M.-I., Vanderdonckt, J. Visual Display Design: a Comparison of Two Methodologies, in Proc. of 1st Int. Conf. on Applied Ergonomics ICAE'96 (Istanbul, 21-24 May 1996), Istanbul - West Lafayette, 1996, pp. 662-667.
21. Leporini, B., Paternò, F., Scorcia, A. Magenta: Flexible tool support for accessibility evaluation. Interacting with Computers 18(5): 869-890 (2006)
22. Microsoft FrontPage 2003, www.microsoft.com/frontpage/
23. Ocawa, Web Accessibility Expert. http://www.ocawa.com/
24. Ratner, J., Grose, E., Forsythe, C.: Characterization and Assessment of HTML Style Guides. In: Proc. of ACM Conference on Human Factors in Computing Systems CHI'96. Vol.2 (1996).
25. Section 508 of the Rehabilitation Act 1973. Available at: http://www.section508.org/
26. Scapin, D., Leulier, C., Vanderdonckt, J., Bastien, C., Farenc, C., Palanque, P., Bastide, R. Towards automated testing of web usability guidelines, in Ph. Kortum, E. Kudzinger (eds.), Proc. of 6th Conf. on Human Factors and the Web HFWeb'2000 (Austin, 19 June 2000), University of Texas, Austin, 2000.
27. Snyder, C. Paper Prototyping: The Fast and Easy Way to Define and Refine User Interfaces. Morgan Kaufmann Publishers, 2003.
28. TAW, Web Accessibility Test. http://www.tawdis.net
29. Vanderdonckt, J., Development Milestones towards a Tool for Working with Guidelines, Interacting with Computers, Vol. 12, No. 2, 1999, pp. 81-118.
30. Vanderdonckt, J., Beirekdar, A. Kwaresmi: Automated Web Evaluation by Guideline Review. J. Web Eng. 4(2): 102-117 (2005).
31. Vanderdonckt, J., Limbourg, Q., Michotte, B., Bouillon, L., Trevisan, D., Florins, M.: UsiXML: a User Interface Description Language for Specifying Multimodal User Interfaces. In: Proc. of W3C Workshop on Multimodal Interaction WMI'2004, Sophia Antipolis, 19-20 July 2004.
32. Vellemann E., Strobbe C., Koch J., Velasco C. A., Snaprud M. (2007). A Unified Web Evaluation Methodology using WCAG. In: Universal Access in HCI (to appear, Proceedings of the 4th International Conference on Universal Access in Human–Computer Interaction, 22—27 July 2007, Beijing, China). New Jersey: Lawrence Erlbaum Associates.
33. WabCluster. The EU Web Accessibility Benchmarking Cluster, Evaluation and benchmarking of Accessibility. Available at: http://www.wabcluster.org/
34. WebXACT. Available at: http://webxact.watchfire.com/
35. Wave 3.0, Web Accessibility Tool. http://wave.webaim.org
36. WebRatio, You think You get. http://www.webratio.com
37. Winckler, M., Xiong, J., Noirhomme-Fraiture, M. Accessibility Legislation and Codes of Practice: an Accessibility Study of Web Sites of French and Belgium Local Administrations. In: proceedings of the 1st International Workshop on Design & Evaluation of e-Government Applications and Services (DEGAS'07) Rio de Janeiro, Brazil, September 11th, 2007. CEUR Workshop Proceedings, ISSN 1613-0073, online at http://ceur-ws.org/Vol-285.
38. World Wide Web Consortium (W3C), Web Accessibility Initiative (WAI). http://www.w3.org/WAI/
39. Xiong, J.; Diouf, M.; Farenc, C.; Winckler, M. Automatic Guidelines Inspection: From Web site Specification to Deployment. In proceedings of 6th International Conference on Computer-Aided Design of User Interfaces CADUI´2006. Bucharest, Romania, June 5-8, 2006.