

Comparing accessibility evaluation tools

A method for tool effectiveness

Giorgio Brajnik*

Dip. di Matematica e Informatica
Università di Udine, Italy
giorgio@dimi.uniud.it

Received: @@@ / Revised version: @@@

Abstract March 2004. This is the pre-final version of the paper published as **Comparing accessibility evaluation tools: a method for tool effectiveness. Universal Access in the Information Society**, 3(3-4), Springer Verlag, pp. 252-263, Oct. 2004; <http://www.springerlink.com/openurl.asp?genre=an>
The published version is slightly changed.

The paper claims that effectiveness of automatic tools for evaluating web site accessibility has to be itself evaluated given the increasingly important role that these tools play. The paper presents a comparison method for a pair of tools that considers correctness, completeness and specificity in supporting the task of assessing the conformance of a web site with respect to established guidelines.

The paper presents data acquired during a case study based on comparing LIFT Machine with Bobby. The data acquired from the case study is used to assess the strengths and weaknesses of the comparison method.

The conclusion is that even though there is room for improvement of the method, it is already capable of providing accurate and reliable conclusions.

1 Introduction

An accessible web site is a site that can be *perceived*, *operated* and *understood* by persons despite their congenital or induced disabilities [14, 19, 16].

I argued elsewhere [3] that web accessibility is just one facet of web *quality of use*, and that quality of use, together with *utility*, *visibility* and *credibility* is one of the pillars upon which the web site success depends. A web site that falls short in any of these properties severely hinders its success.

* Scientific advisor for UsableNet Inc., manufacturer of LIFT Machine, one of the tools used in the case study reported by the paper.

One of the claims in [3] is that unless automatic web testing tools are deployed in the normal processes of design, development and maintenance of web sites, the quality of web sites is unlikely to improve. This is due to an unfavorable combination of factors related with the dynamics of these processes and the nature of web sites. Very short release cycles, lack of resources (time, persons, money), incomplete and vague specifications, rapidly evolving technologies, complexity of designing information and interaction architectures, ease of use of powerful authoring tools (like Dreamweaver or Flash) are the main reasons why web developers do not pay too much attention to the quality level of web sites. Only if (at least some) quality factors are treated in an automatic way, so that the computer deals with the most repetitive and trivial details, can the developer devote time to learn and focus on more important quality issues.

At the moment there are several tools for testing accessibility of web sites (see an appropriate page at W3C [22]). They differ in several dimensions, ranging from functionalities (testing vs. fixing), to supported interaction form (online service vs. desktop application integrated in authoring tools), effectiveness, reliability, cost, etc. Some of the tools have been in the arena for several years (for example Bobby, that was initially developed by CAST and freely available; now it has been acquired by a commercial company and it is being sold).

It is very important to be able to evaluate the quality of these tools, as they play the key role of enabling an *average* web developer to develop better web sites. Only if reasonable evaluation methods are defined and used the quality of these tools can be assessed in a relatively standard way. These evaluations will improve the ease with which a web developer could compare different tools and perform the appropriate choice. Furthermore these evaluations could stiffen the competition between tool manufacturers, and in the end improve the tools themselves.

The goal of this paper is to illustrate a method for comparing different tools that is *useful* to pinpoint strengths and weaknesses of tools in terms of their effectiveness, *viable* in the sense that the method can be applied with limited resources, and *repeatable* in the sense that independent applications of the method to the same tools should lead to similar results.

A *useful* evaluation method is capable of producing results that are relevant to somebody for something. In our case, the method should be able to clearly identify differences in effectiveness of testing tools so that potential users can choose in an informed way the tool that best suit them.

A *viable* evaluation method is not overly demanding of resources (time, money, persons, skills, infrastructure). In our case at most a few persons should be involved in carrying it out, without having to resort to additional resources, and in a matter of a few days.

Finally, a *repeatable* evaluation method is such that repeated applications of it at different times, on different web pages, and with different evaluators lead to the same results.

These properties of the method are partly demonstrated by results derived from a case study using LIFT Machine (a multi-user web accessibility testing system developed by Usablenet — version 1.4) and Bobby (a desktop application for test-

ing web accessibility acquired and engineered by Watchfire — version 4, “Bobby Core WorldWide”).¹

This paper is a long version of [4].

2 Related work

A thorough analysis was conducted by Thatcher [18]. Unfortunately such an evaluation was only temporarily posted on his web site before being taken off-line. His evaluation (of 6 accessibility testing tools) was aimed at determining the cost/benefit ratio and helping potential customers to select the most appropriate tool. In addition to considering costs, availability, and accuracy, the evaluation scope was *quality of use* in general. The method he used is heavily based on manual and systematic inspection of the results produced by the tools on selected test pages, and is therefore less generally applicable than the method proposed in this paper, as it requires carefully prepared test files and a long and subjective analysis of the results.

A survey of the web usability properties that are (or could be) tested automatically has been illustrated in [1] and others have used some of these tools to survey the web [17]. A CHI workshop was devoted to discuss and compare the capabilities of guideline-based tools and model-based tools [5]. However the problem of designing an appropriate method for assessing effectiveness of these tools was tackled in neither of these cases.

Ivory and Hearst propose in [11] a taxonomy of usability evaluation methods. The taxonomy is very general and encompasses a wide range of methods including manual and fully-automatic ones. However the taxonomy does not cover the evaluation of evaluation methods and tools, which is what this paper aims at.

In a more recent paper [12], Ivory and her colleagues aimed at evaluating quality of use of testing tools. They performed an experiment where web designers were asked to use testing tools and to modify web sites accordingly to what tools suggested. Then, in a second experiment, the authors determined how effective such changes were for disabled web site visitors.

The aim of this paper is less ambitious: rather than capturing overall quality of use of tools and their consequences on tested web sites, this paper focuses on a much more restricted definition of *tool effectiveness* (see section 3). The consequences are twofold: on the one hand the results that can be obtained by following the method proposed by this paper are more specific and more limited than those obtainable by following Ivory’s et al. method. On the other hand, however, it is less likely to produce invalid data. In fact, the method described in [12] raises a number of issues that are not addressed by the authors and that show how difficult it is to carry out a sound usability evaluation of these tools:

¹ Comparing these tools is a little unfair given their scope, flexibility, power and price: LIFT Machine is targeted to an enterprise-level quality assurance team and whose price starts at \$6000; Bobby 4.0 was available for free (now it runs at about \$300) and is targeted to a single individual wanting to test a relatively limited number of pages. Nevertheless the comparison is useful as a case study for demonstrating the evaluation method itself.

1. The web designers skill level in web accessibility is not known. Yet skill level of the participants in the experiment is a key variable affecting how effectiveness, usability and quality of use of testing tools are measured.
2. How designers assessed accessibility of tested web sites is not known. And yet many results reported by the paper depend on such a method. Different assessment methods lead to different results. For example, a *sniff test* is carried out by an assessor with no reference to any principle or guideline, whereas in a *conformance assessment* assessors use a checklist of guidelines [23], and in a *usability testing* assessors observe the behavior of actual users. These three types of methods produce widely different results. And even though only the conformance assessment method is used, but with different assessors or with a different set of reference guidelines, it is very likely to yield different results when applied in these different settings.
3. Web designers were allowed to use these tools for no more than 20 minutes, which appears to be a very short time (i) to get acquainted with potentially unknown tools, (ii) to carry out a function that might itself be partially unknown (i.e. assessing accessibility conformance) and (iii) to follow tools suggestions to fix web sites web designers did not create.
4. The paper mixes two different tasks: assessing conformance of web sites with respect to certain guidelines and evaluating usability of web sites for certain disabled people. Tools were used to perform the former task, but they were evaluated with respect to the latter.

These issues are evidence that a sound evaluation of evaluation tools is a difficult task because it involves many aspects that may affect the results, including usability of tools, usability of resulting web sites and relevance of selected guidelines. For this reason, the current paper focuses only on a very restricted definition of tool effectiveness (see section 3), which is a necessary property for characterizing a more general quality of use of tools.

A tool validation method is proposed in [2], that is based on the indirect feedback that a web developer provides to a testing tool. The testing tool can detect (within certain limits) if two consecutive evaluations of the same web site generate the same sets of problems. If a problem was found in the first evaluation, but it has not occurred in the second one, it is assumed that the web developer has fixed that problem. And that the fixing has been prompted, suggested, guided by the warnings issued by the tool. If this is the case then the number of problems that do not occur in a second evaluation of the same web site is an indirect measure of the utility of the tool. The biggest limit of this method is that it is based on the behavior of the web developer who might decide to fix a problem only because the tool suggested to do it, rather than because it is recognized as being an accessibility issue.

Another more general and abstract view of how web testing tools can be evaluated is presented in [15].

3 Relevant questions

In this paper I limit the discussion on tool effectiveness only with reference to the task of *fault identification and diagnosis* in the context of a conformance assessment, i.e. the task of supporting the web developer in identifying violations of selected guidelines (fault identification – detecting the symptoms) and in associating them to possible causes (diagnosis – finding the defects). Other functions that the tools may provide, like support in fixing the defects (e.g. [21]), will not be considered.

Furthermore, only tools that can perform fault identification and that can diagnose faults can be evaluated by the method described by this paper. For example, previewing tools (like [7]) cannot be evaluated since they do not *explicitly* identify faults and their causes.

Effectiveness of tools can be framed around these basic concepts:

- how *complete* is a tool in covering the relevant aspects to be detected,
- how *correct* is it, and
- how *specific* is it.

Completeness and correctness are both necessary for characterizing effectiveness. A complete (but incorrect) tool could flag every page with all sorts of problems, generating therefore a large number of *false positives*, i.e. statements of detected problems that are not true. Conversely, an incomplete but correct tool could issue a problem if and only if an IMG tag has no ALT attribute. No false positives are generated by such a tool, but there are many other possible accessibility defects that get undetected. Such a tool generates a large number of *false negatives*: true problems that are not flagged.

The consequence of false positives is that evaluation tools generate noise, that disturbs and annoys users, leading to under-use of these tools and to occasional errors (due to not seeing some of the warnings). People usually cope with false positives by customizing the tools, for example by enabling only certain types of tests or by defining specialized tests that are based on specific conventions adopted within the web site being tested (like detecting spacer images by knowing that the image file is named `sp.gif`).

False negatives have worse consequences: tools fail to warn the user about potential problems, that can go un-noticed. Unfortunately there's no way to cope with false negatives other than deploying additional methods for testing (for example, human inspection or user testing).

In addition to completeness and correctness, I introduce also specificity. The specificity of a tool reflects in the level of detail that the tool is able to use when describing a potential problem. If a tool raises only very general concerns about a page (for example, it warns the user with a message like “the page contains non-textual elements with no text equivalent”) then the specificity of the warnings that are issued is too little with respect to the amount of details that is needed by the web developer to understand the problem, diagnose it (i.e. to find the root causes) and to plan its solution. A tool suggesting that an image does not have an ALT and that the correct ALT should be the empty string (i.e. `ALT=""`) because the image

is a spacer, is more useful than a tool simply saying that the image requires an ALT.

As discussed above (in section 2, Related work, and at the beginning of this section), the method focuses on a specific view of *tool effectiveness*. In particular it does not consider how easy it is to learn to use a tool, how easy it is to recall what the tool does and how it works, how good it is in supporting the repair task, how good it is in providing relevant suggestions at the right moment, and how supportive it is in accessibility training. Although these are all important aspects I believe they are all dependent on effectiveness as defined above. It would be no use to run usability experiments aimed at determining these properties if a tool fails to be effective.

3.1 Completeness of a tool

Completeness is a measure of how many accessibility defects present in the web site are caught and correctly shown to the user. Completeness is related to how well the tool reduces false negatives.

Completeness is a difficult property to characterize operationally. In fact it requires to know the true problems in advance. Accessibility is an *external* quality property, i.e. it is determined not only by the web site in itself — its HTML, Javascript, CSS, or Flash code; the web server behavior — but also by the interaction with the web visitor, his/her own context, situation, and cognitive state. Therefore determining the true problems means that accurate usability investigations (through user testing or heuristic evaluations) need to be performed.

In order to yield a viable method, our stance is to base the definition of completeness of a tool with respect to conformance to a set of accepted accessibility guidelines, namely the *Web Content Accessibility Guidelines 1.0* defined by W3C/WAI [24]. Even though conformance to these guidelines does not guarantee that a web site is indeed accessible (see, for example, [13] for results derived via user testing performed with disabled persons), the guidelines are a reasonably good starting point for evaluating completeness of testing tools, since conformance to such guidelines is viewed by many as a necessary condition for accessibility.

3.2 Correctness of a tool

Correctness is the proportion of problems reported by the tool that are indeed true problems. That is, correctness is related to how well a tool reduces false positives.

False positives cannot be avoided for accessibility (unless we set for a very low level of completeness). In fact many accessibility problems deal with perception and interpretation of information, and in few cases these aspects can be made fully explicit and formally characterizable. For example, the guideline that says “14: Ensure that documents are clear and simple” obviously cannot be tested automatically. In these cases tools can use some model of text readability (e.g. the Gunning–Fog measure [10]) and support the decision–making process of the user who has to determine if the page uses a simple enough language. The role of the

tool in this case is to highlight certain features of the web site so that an appropriate decision can be performed by the tool user upon further investigation.

This is the reason why many tools differentiate between *automatic* and *manual* tests. Automatic tests flag only features that are true problems, while manual tests flag features that are potential problems that cannot be automatically assessed.²

The ratio of manual tests with respect to the available tests within a tool is one way to characterize the correctness: a tool based only on manual tests would raise many issues, many of them could have been filtered away without bothering the user, i.e. are false positives.

3.3 Specificity of a tool

Specificity of a tool is defined as the number of *different* possible issues that can be detected and described by a tool. The larger this set (given a certain level of completeness) the more capable is the tool of providing specific warnings and suggestions, and therefore the more useful it is for the web developer.

Specificity of a tool is not an easy property to be determined and not necessarily related to the tool effectiveness. For example if tool A flags INPUT elements that are not properly labeled and tool B has five different tests, each flagging respectively text input, radio buttons, check boxes, select menus and text areas, then B is not necessarily better than A. To determine that B is better than A you need to run specific usability tests and see which of the tools provides more contextual and helpful suggestions. This goes beyond the scope of the method proposed in this paper, that should require limited effort and time to be carried out. For this reason I assume that the more specific a tool is the more effective it is.

4 Relevant metrics

Appropriate *metrics* need to be defined in order to adequately characterize completeness, correctness and specificity. The metrics discussed below are not usability metrics, as we are not interested (at least for the scope of the present paper) in evaluating usability of testing tools.

A metrics, or measurement method [9], should include at least a description of how:

² More specific types of results could be considered. For example a distinction among *definite errors*, *probable errors*, *manual warnings triggered by content*, *untriggered manual warnings* would yield a finer grid serving as a base for a richer evaluation of tools. Some of the testing tools provide these finer distinctions. However it may be difficult to classify the tool output according to those categories (this information might not be always available from the tool output) and if two tools provide different types of results, it will be difficult to compare them. For this reason the method is based on two types of results: those that the tool assumes to be true problems and those that are warnings.

The consequence is that the evaluation method is blind with respect to these finer distinctions: tools that provide these intermediate warnings are treated in the same way as tools that provide manual warnings.

1. data have to be elicited,
2. data are identified,
3. data are categorized,
4. results are computed.

The metrics that we need should be able to elicit, identify, categorize and summarize data that can provide answers to the questions discussed in section 3.

Since operational definitions of completeness are difficult (they require to know the set of possible defects of a web site in advance), an approximate concept will be adopted.

The method is to be used in comparative evaluations of tools (i.e. to compare two different tools, or to compare two different versions of the same tool). In order to operationally define the concept of completeness, the method assumes that the set of *true problems* of a web site is equal to the set of *detected true problems*. The latter is obtained by running both tools on the same web site, computing the union of the sets of problems produced by them, and removing all those that upon manual investigation turn out to be false positives. It is also assumed that *manual investigation* means review performed by an accessibility expert using the guidelines as a reference.

4.1 Data acquisition procedure

4.1.1 Tested sites The tools being compared by the method should be applied to a number of different web sites. Rather than developing ad-hoc test pages, and then running the tools against these test pages, I decided to aim at real-world web sites.

Test pages are difficult to create starting from relatively high level requirements like WCAG. It is likely that some aspects are not represented in test pages, or that only certain solutions are considered. Well done test files need to be: (i) complete (i.e. they should contain instances of all the possible guidelines violations and examples of good practice of use of available web technologies – HTML, JavaScript, CSS, Flash, SMIL, etc.) and (ii) realistic, i.e. they should contain instances of violations that can actually occur in the real world. Spending resources in building test files for focusing on accessibility issues of *ASCII art* (i.e. emoticons, symbols like ==>) seems an overkill. In addition, proceeding top-down (from abstract guidelines to checkpoints and specific examples) brings the risk of spending a lot of resources on test files for checkpoints that the selected tools do not implement in such a detail (for example, selected tools may not have automatic tests for processing pairs of background/foreground colors: the time spent in developing detailed test cases is not productive).

Unless test files are complete and realistic they cannot be used to assess effectiveness of tools. In addition, test files need to be maintained, corrected and updated.

For these reasons only real web sites are considered in the case study. Consider however that the method can be applied to test pages, if available. The advantage

in such a case would be that detecting false negatives would be simpler, as the set of true problems is known in advance.

In order to stress the tools and force them to generate as many false positives as possible, web sites that are known (or presumed) to be accessible have to be used. In addition to these sites, a number of other (not known to be accessible) web sites are tested with the aim of stressing the ability of the tool to avoid false negatives. These sites might include public and well known news portals and large corporate web sites.

The following web sites have been used in the case study: w3.org/WAI (Web Accessibility Initiative of W3C), jimthatcher.com (one of the leading experts in accessibility), www.tsvbi.edu (Texas School for the Blind and Visually Impaired), www.webaim.org (an organization involved in accessibility), www.nfb.org (National Federation for the Blind), www.trace.edu (TRACE research center on disabilities), www.aa.com (American Airlines).

In order to reduce the number of pages to be analyzed, only some of the pages of these web sites have been selected. And to avoid that (dynamic) pages change between the evaluation performed by one tool and the one performed by the other tool the following precaution was taken.

Each web site was spidered using `wget` (a freely available web spider running on linuxes) and all pages within a distance of 2 links from the home page were downloaded and URLs were appropriately translated into relative ones so that the downloaded site would be still navigable. Then, the resulting files and directories were loaded on a web server used for this experiment. Each tool then had to crawl these mirrored web sites. In this way different runs of the tools would examine exactly the same pages regardless of cookies or of actual HTML content delivered by the web server. No distinction was made for possible *text-only pages* that might have been available: they were treated as all the other pages.

To further reduce the amount of pages to be analyzed by the tools a hard limit of maximum 50 pages was adopted. With LIFT Machine this was easy to set, whereas with Bobby it was impossible to enforce such a limit, and I had to enforce only the `depth=2` constraint (only pages that where at a distance of 2 or less from the starting page).³

Each tool was instructed to use the AA conformance level (i.e. applying tests implementing checkpoints having priority 1 and 2). This is justified by the fact that many consider AA to be the highest official level of conformance to WCAG guidelines that can be practically achieved. For tools that allow a higher level of customization (for example LIFT Machine allows a fine definition of parameters affecting the way in which a test behaves) the default setting was used.

4.1.2 Data collection Each run of the tool ended by generating an HTML and XML report of all the issues raised by the tool on the tested web site. An *issue*

³ I recommend to use the same limits for both systems; otherwise the consequence is that there might be several pages that are tested by one tool only, reducing thus the effectiveness of the comparison method, since the issues associated to these pages are excluded from any further analysis. This has happened in the case study, due to differences in the crawling methods adopted by different tools.

	n. of generated issues	n. of selected issues
Bobby	21523	93
LIFT	24884	212
total	46407	305

Table 1 Summary of the selected sample of issues

is an instance of a potential problem detected by a tool. For example the test that checks existence of the ALT attribute for IMG may generate 5 issues on a page: one for each IMG missing the ALT.

A data reduction step was needed to reduce the number of issues to be considered, since a crucial step of the method requires manual inspection of the issues to determine false positives and false negatives.

This reduction of size was performed by randomly selecting 5 issues for each checkpoint. In practice the XML files were processed (via a simple XSLT stylesheet) to produce a list of lines, one line per issue. Issues generated by the two tools were then merged, sorted by checkpoint, and then 5 issues per checkpoint were randomly chosen. Table 1 shows the summary of the sample size. The total number of issues generated by the two tools exceeds 46000; if the method would be based on systematic manual inspection then evaluators would be required to manually inspect all of them. This is the reason why a sampling procedure was used to reduce the data volume by a factor greater than 150.

Such a sampling procedure guarantees random selections of issues belonging to a checkpoint. Since the selection is repeated for all the checkpoints, any checkpoint for which there is at least one issue has the opportunity to appear in the resulting set of selected issues. A drawback of this selection procedure is that the sample of issues for a given checkpoint may not be representative of the actual number of issues. For checkpoints where there is a single test which fires on mostly every page (e.g. “If you use colors to convey information, make sure the information is also represented in another way”) then all the selected issues for that checkpoint will refer to that test only. On the other hand checkpoints that are implemented by a larger number of tests (e.g. “1.1: Provide a text equivalent for every non-text element”) will result in a set of issues that are not representative of such a variability.

Each issue was associated with the tool that generated it, to the page URL on which it appeared, the line number, to the test description/name, to a manual/automatic attribute, and to the checkpoint the test refers to.

The manual/automatic attribute describes the kind of diagnosis that the test performed. For Bobby *manual* issue means an error instance with support belonging to *Partial*, *PartialOnce*, *AskOnce*⁴; for LIFT *manual* issue means an issue with type *manual*.

⁴ This is Bobby’s terminology corresponding to what we earlier referred to as *manual warning triggered by content* (for *Partial* or *PartialOnce*) and *untriggered manual warning* (for *AskOnce*).

Data in table 1 show that the proportion of issues generated by the two tools differ in the selected issues vs. generated issues. This is due to the sampling procedure and the way in which issues are grouped into checkpoints by either tools. For example LIFT Machine generates many more issues within the WCAG 1.1 checkpoint than Bobby, and therefore it is likely that the selected sample contains LIFT Machine generated issues rather than Bobby ones. While this bias does not affect correctness claims, it does affect the claims about false negatives, as further discussed in section 4.3.

4.2 Issue classification

The resulting set of issues has to be manually inspected to determine which issue can be categorized as a false positive or false negative, for both tools. The following criteria were followed:

- **False positive (FP)**: an issue generated by a tool was classified as a FP for either tool if, upon investigation of the HTML page the issue refers to, a human inspector would consider the issue irrelevant or wrong. For example an issue like “Nest headings properly” was classified as a FP on a page where no heading tags (i.e. H1, H2, ... H6) are used or where headings were used properly. A FP would also include those cases where the issue is plainly wrong: “Avoid using structural markup for visual formatting” on a page that correctly uses TH for marking up data table headers.

This classification is based on a *conformance* criterion, rather than *true accessibility*. This is justified by the fact that users of evaluation tools expect the tool to highlight potential problems, suspect features in the pages that need to be examined in a deeper detail.

No difference in interpretation occurred between manual and automatic tests.

Notice that an issue could result in a FP for the tool that generated it, or also for the other tool, if a similar issue were generated.

- **False negative (FN)**: an issue X generated by tool A, not classified as a FP, is used as a reference for tool B: if B does not generate any issue that is equivalent to X, then X is classified as a FN for tool B (B missed it). Again manual and automatic tests were treated in the same way. For example “Do not use the same link phrase more than once when the link points to different URLs” is an issue that was systematically classified as a FN for LIFT since LIFT has no such test. On the other hand “Inform users if pop-up windows appear (the script opens new windows)” was a FN for Bobby (for a similar reason).
- **OK**: otherwise. An issue generated by tool A was marked as OK for A if it is not a FP; it was marked OK for tool B if it was not a FN nor a FP. If tool A generates a very specific issue X (like “Spacer image has no valid ALT”) and tool B generates a more general issue that covers the same problem (like “Provide ALT to images”), then X is not classified as a FN, but as an OK issue for tool B (since B catches it, even though at a more general way).

In the end an issue is classified twice: once (as *OK* or *FP*) with respect to the tool that generated it, and once (as *OK*, *FN* or *FP*) with respect to the other tool. For

	automatic only			automatic or manual		
	FP	FN	total	FP	FN	total
Bobby	20	38	39	78	60	93
LIFT	5	16	110	50	21	212

Table 2 Summary of the classified issues

example a hypothetical issue of the test “Inform users if pop-up windows appear (the script opens new windows)” raised by LIFT could turn out to be a FP. In such a case the issue would be labeled as FP for LIFT and, if Bobby would have raised it as well, it would be labeled FP for Bobby too. On the other hand, if Bobby would not have arisen that issue, the issue would be labeled OK for Bobby. If instead that issue were not a FP, then it would be labeled OK for LIFT and, if Bobby raised it, OK for Bobby. If Bobby missed it, then it would be a FN for Bobby.

Notice that if both tools missed a defect, then in no way the evaluation method is capable of detecting this. Additional means for testing tools completeness are needed to avoid this, like using predefined test cases, as discussed in section 4.1.1.

Table 2 shows the resulting numbers. Out of the 39 issues generated by automatic tests of Bobby, 20 were classified as FP and out of the 93 issues generated by automatic or manual tests of Bobby, 78 were FP. And out of the overall total of (automatic) issues considered ($149 = 39 + 110$), 38 were considered FN for Bobby; and out of 305 ($93 + 212$) automatic and manual issues, 60 were considered FN for Bobby. And similarly for LIFT.

4.3 Completeness parameters

Two strategies are adopted in the definition of completeness parameters.

- **intensional aspects:** the idea is to see how well the tools nominally cover each checkpoint of the guidelines. Parameters defined with this criterion include:
 - *number of checkpoints* that are covered by the tool (i.e. if the tool can signal violations of such a checkpoint).
This parameter is related to completeness, since the fewer the checkpoints covered by the tool the less likely is it that the tool detects conformance violations.
 - *fraction of tests that are automatic.*
This is important as automatic tests are the most useful feature for the power user of the tool, who needs to enhance his/her productivity in testing a web site when determining the status of a site. The larger this number the more complete is the task performed by the tool.
- **extensional aspects:** i.e. based on the results produced by execution of the tools. These parameters include:
 - proportion of *detected true problems* that are missed by the tool and are labeled as *automatic*;
 - proportion of *detected true problems* that are missed by the tool and are labeled as *manual* or *automatic*.

	covered checkpoints	n. of tests		
		automatic	manual	total
Bobby	46/46	21	49	70
LIFT	45/46	50	53	103

Table 3 Summary of the tests used by the tools

	automatic only			automatic or manual		
	total	FN	proportion	total	FN	proportion
Bobby	149	38	26%	305	60	20%
LIFT	149	16	11%	305	21	7%

Table 4 Summary of the false negatives generated by the tools

In the case study, intensional values for Bobby and LIFT are shown in table 3.

LIFT does not cover one checkpoint (11.4: “If, after best efforts, you cannot create an accessible page, provide a link to an alternative page that uses W3C technologies, is accessible, has equivalent information (or functionality), and is updated as often as the inaccessible (original) page.”). And approximately 49% percent of LIFT tests (50 out of 103) are automatic compared with 30% of Bobby tests that are automatic (21 out of 70).

For the extensional aspects, the case study yielded the data shown in table 4.

Out of 149 automatic issues that were classified, 38 turned out to be FN for Bobby and 16 for LIFT. And out of the 305 automatic and manual issues, 60 were FN for Bobby and 21 for LIFT.⁵

4.4 Specificity parameters

The parameters for specificity include:

- *number of different tests implementing each checkpoint.*

Each test signals violation of a certain checkpoint. There might be tests that are more specific: for example, checkpoint WCAG 1.1 does not tell apart missing ALTs for a spacer from missing ALTs of button images, but a tool may provide these finer distinctions. And since checkpoints are rather independent from each other, very seldom a test signals violation of more than one checkpoint.

This parameter is related to specificity: the more tests are available for each checkpoint the better it is as the user receives potentially more specific warnings and suggestions.

⁵ This is a case where the values reported in table 1 affect the FN percentages. In particular, since FN for Bobby is defined in reference to the behavior of LIFT, if we consider a larger number of issues generated by LIFT then there are more chances that we find a FN for Bobby. Therefore FN for Bobby is correct, while FN for LIFT is underestimated.

WCAG checkpoint	Bobby		LIFT	
	n. of tests	automatic tests	n. of tests	automatic tests
1.1	12	6	43	19
13.1	4	1	2	1
5.1	1	0	4	3
6.5	1	1	3	3
3.4	1	1	3	3

Table 5 Coverage of selected checkpoints (those with the largest number of tests)

	automatic only			automatic or manual		
	total	FP	proportion	total	FP	proportion
Bobby	149	20	13%	305	78	26%
LIFT	149	5	3%	305	50	16%

Table 6 Summary of the false positives generated by the tools

Table 3 shows that LIFT has about 47% more tests than Bobby (103 rather than 70). And table 5 shows selected checkpoints and the number of tests that implement them (only checkpoints implemented by at least 4 tests are considered in the table). Bobby has 12 tests that implement checkpoint 1.1 — *Provide a text equivalent for every non-text element*, 6 of which are automatic. LIFT has 43 tests for the same checkpoint (thanks to the finer classification of image types performed by LIFT), 19 of which are automatic. These numbers are exceptionally high, as the subsequent checkpoints are implemented by 5 or fewer tests.

4.5 Correctness parameters

The parameters for correctness are:

- the proportion of manual and automatic problems reported by the tool that are *detected true problems*. However, for uniformity with the parameters used for completeness, we consider here the proportion of the issues reported by either tool that are flagged as FP for the tool being considered.
- the proportion of automatic problems reported by the tool that are *detected true problems*. In this case, too, we consider the proportion of the issues generated by either tool that are flagged as FP for the tool being considered.

Table 6 shows the values gathered for these parameters. Out of 149 automatic issues that were analyzed, 20 were found to be FP for Bobby (of the 39 that were generated by Bobby alone — see table 2).

Out of 149 automatic issues, 20 were FP for Bobby and 5 for LIFT. And out of 305 manual or automatic issues, there were 78 FP for Bobby and 50 for LIFT.

	automatic only		automatic or manual	
	fp	fp confidence interval	fp	fp confidence interval
Bobby	13%	(6%, 21%)	26%	(19%, 32%)
LIFT	3%	(0, 7%)	16%	(11%, 22%)
	fn	fn confidence interval	fn	fn confidence interval
Bobby	26%	(16%, 35%)	20%	(14%, 26%)
LIFT	11%	(4%, 17%)	7%	(3%, 11%)

Table 7 Confidence intervals ($\alpha = 0.01$) around the proportions of FP and FN for the tools

4.6 Data analysis

The data collected and classified as described in previous sections were statistically analyzed to produce appropriate conclusions.

The comparison method can yield the following statistical types of conclusions:

- confidence intervals around the proportions FP and FN (for the two tools, and by considering either all the issues or the automatic ones only) based on a significance level of $\alpha = 0.01$ ⁶. Confidence intervals are needed to be able to state the level of completeness or of correctness of a tool.
- statistical support for the claims that *the proportion fp of tool A is less than the proportion fp of tool B* and that *the proportion fn of tool A is less than the proportion fn of tool B*. The first claim will be called $HFP_a : fp(A) < fp(B)$, and the second one $HFN_a : fn(A) < fn(B)$. The corresponding null hypotheses are $HFP_0 : fp(A) = fp(B)$ and $HFN_0 : fn(A) = fn(B)$. For both claims the method provides the p -value, i.e. the probability that the claim HFP_a or HFN_a is false.⁷
- in addition, to estimate the consequences of possible issue classification errors (i.e. incorrectly labeling an issue as OK, FP or FN), the method produces also the range of variations of the proportions fp and fn so that the claims HFN_a and HFP_a are valid with a p -value of 0.01.

In the case study the confidence intervals (with $\alpha = 0.01$) are described in table 7.

This means, for example, that based on the data gathered by the case study, with probability 99% Bobby generates between 6% and 21% FP (for automatic tests only) and between 19% and 32% FP when automatic and manual tests are considered. For LIFT the values are less than 7% and between 11% and 22% respectively.

⁶ A confidence interval of a parameter around a value and with a given significance level α describes the possible variability of the parameter when a different sample of data is analyzed. α gives the probability that the parameter stays within the interval.

⁷ For example, the claim HFP_a valid with probability 0.01, means that the data gathered in this experiment in 99 cases out of 100 support the claim that A produces less FP than B.

claim	automatic only	automatic or manual
	<i>p</i> -value	<i>p</i> -value
HFP_a	0.09%	2.6%
HFN_a	0.05%	< 0.01%

Table 8 *P*-values for the two comparison claims

The reason for this variability in the results stems from the statistical variability of the sample of data being acquired, collected and classified. Pure chance, bias due to the web sites being considered, errors in collecting the data and errors in classifying them may lead to different results. But with probability 99% another experiment like this will yield a value for fp and fn that stays within the intervals shown in the table.

It is reasonable that the percentages increase when moving from the *automatic-only* to the *automatic or manual* scenario since adding more tests, that are also less focused, increases the chances of generating FP.

The table shows also that Bobby yields between 16% to 35% FN, i.e. it misses between 16% to 35% of true problems, when automatic tests are considered, and between 14% to 26% when automatic and manual tests are considered. For LIFT Machine the values are 4% — 17% and 3% — 11%.

In this case moving from the *automatic-only* to the *automatic or manual* scenario decreases the percentages. This is due to two reasons:

1. adding more tests means that less properties remain unchecked
2. due to the way in which the *detected true problems* are defined, by adding more tests to tool A we also increase the potential number of FN of tool B (since FN for tool B use the issues generated by A as a reference).

Table 8 shows the *p*-values for the two comparison claims $HFP_a : fp(LIFT) < fp(Bobby)$ and $HFN_a : fn(LIFT) < fn(Bobby)$.

The data strongly suggest that LIFT produces less FP than Bobby (with probability 0.09%, i.e. 9 out of 10,000) that this is not the case if automatic tests are considered, or 2.6% if automatic and manual test are considered), and that LIFT produces less FN than Bobby (where probabilities are 0.05% and less than .01% respectively).

Finally, what is the sensitivity of the method to classification errors? In other words, which is a data classification error margin that does not force us to reject a claim?

If we were to accept a *p*-value of 5%, then table 9 gives the percentage and absolute values of the error around the proportion fp and fn that would make the claims still valid.

The results shown by table 9 say that with probability 95% the claim HFP_a is true even if we misclassify up to 7 issues if automatic tests are considered and up to 11 issues otherwise. For the other claim the ranges are 11 and 25 respectively.

claim	automatic only		automatic or manual	
	error	abs. range	error	abs. range
HFP_a	$\pm 2.4\%$	7	$\pm 1.9\%$	11
HFN_a	$\pm 3.7\%$	11	$\pm 4.1\%$	25

Table 9 Error margins

4.7 Results of the case study

If it weren't for the underestimation problem discussed in section 4.3, the method applied to this case study would support the following conclusions:

Correctness

- with probability 99% LIFT Machine generates less than 7% of false positives when using automatic tests only (less than 22% otherwise)
- with probability 99% Bobby generates less than 21% of false positives when using automatic tests only (less than 32% otherwise)
- the claim that LIFT Machine generates less false positives than Bobby is true with probability 99.91% if we consider only automatic tests, and 97.4% otherwise.

Completeness

- Bobby covers all the 46 checkpoints while LIFT Machine covers 45 of them
- 30% of Bobby's tests are automatic, while 49% of LIFT Machine tests are automatic
- with probability 99% LIFT Machine is affected by less than 17% of false negatives when using automatic tests only (less than 11% otherwise), but consider the underestimation problem discussed in section 4.3
- with probability 99% Bobby is affected by less than 35% of false negatives when using automatic tests only (less than 26% otherwise)
- the claim that LIFT Machine generates less false negatives than Bobby is true with probability 99.95% if we consider only automatic tests, and more than 99.99% otherwise. But consider the underestimation problem discussed in section 4.3.

Specificity

- Bobby implements 70 tests while LIFT Machine has 103 tests
- for the 6 most populated checkpoints (i.e. those with the largest number of tests that implement them) LIFT Machine has a number of automatic tests that is equal or greater than the corresponding number for Bobby (up to 50% more tests for some of the checkpoints)
- for the 6 most populated checkpoints LIFT Machine has a number of (automatic and manual) tests that is equal or greater than the corresponding number for Bobby (up to 350% more tests for some of the checkpoints).

5 Conclusions

The comparison method described in the paper is viable, it is capable of producing precise results concerned with correctness, completeness and specificity of the tools being compared, it is capable of supporting direct comparison statements. It can therefore be effectively used to compare a pair of tools.

Some experimental data can be gathered to estimate the consequences of classification errors, which in turn gives an idea of how reliable a comparison is. And therefore of how much statistically sound the results are, with respect to the statistical variation of the data being sampled and classified.

Validity of the method (i.e. how much the results that it yields are true) cannot be proved on the basis of a single case study. However the following arguments can be used as (weak) positive evidence:

- the method itself can provide some data about possible classification errors tolerance. In our case table 9 shows such a tolerance.
- Secondly, some of the results shown above (namely confidence intervals for fp and fn shown in table 7) are based on a value for α that is rather small, leading to conclusions that are conservative. With higher values for α confidence intervals will shrink, and the tolerance to classification errors will increase.
- Thirdly, qualitative considerations of the tools compared in this case study agree with the results produced by the method, namely that LIFT Machine is more correct, more complete and more specific than Bobby. LIFT Machine v. 1.4 is (i) more recent than Bobby 4, it is (ii) a high performance server system that costs roughly 20 times more than Bobby, and it is (iii) based on specific technology (called Semantic Analyzer, see [20]) whose aim is to reduce the number of false positives by guessing the role played in HTML pages by elements like images, tables and text in forms and thus allowing the LIFT Machine to produce more focused issues. These three considerations agree with what the data say, and we use this agreement as an informal and weak way for assessing validity of the method.

Therefore we (tentatively) conclude that the method is also useful.

If we were to redo the same evaluation, a person would need probably no more than 2 full-time days, from selecting and downloading web sites to performing statistical analysis of collected data. Therefore we conclude that the method is also viable.

On the other hand the method has some limits.

- Classification of issues is a subjective activity, and as such its outcome can change if different persons do it, or if the same person does it at different times or in different situations. The only way to cope with this is to run the method in a number of persons (evaluators), each performing classification of the same data, and thinking of a way for resolving conflicting classifications (like a post-classification discussion).
- It is somewhat problematic and subjective to determine if a tool applied to a page generates an issue that is equivalent to an issue generated by the other tool. Yet this type of decision is needed when classifying issues. The problem

here lies in the fact that different tools implement different tests, possibly also at different level of detail. To make the method more robust these evaluations should be carried out by a pool of evaluators and with a voting mechanism for determining the proper classification of an issue.

- The sampling procedure is suboptimal because it does not generate a sample of issues that is representative of the whole set of issues. In addition it does not yield a sample where the proportion of issues generated by the tools is the same as the overall proportion of issues. It probably has to do with the way in which issues are grouped into checkpoints by each of the tools. As discussed above (section 4.3), this overestimates the FN for a tool and underestimates the FN for the other tool.
- The method requires that tests are classified as manual or as automatic and this is not always easy to do. In the case of LIFT Machine, its GUI shows clearly if a test is automatic or not; Bobby doesn't. In the case study the type of test for Bobby was induced by the issues that have been generated by the tests, and for the few that were not used during the runs the test title was used as a hint.
- The method could be based on a finer distinction of the type of warnings that a tool provides rather than just manual vs. automatic (for example, content triggered warnings and untriggered warnings; or warnings associated to a certainty factor representing how much certain an assertion is). While being useful for the comparison, such finer distinctions would require a more complex way for categorizing the data produced by tools, and so far I haven't been able to go beyond the manual vs. automatic distinction.
- The method requires the tools to generate XML files for their results; these files need then to be further processed to support the sampling procedure. In the case study it was relatively easy to create a single XSLT stylesheet able to merge the XML files produced by LIFT Machine and by Bobby. For other tools the situation might be more difficult. Use of EARL (Evaluation and Report Language — an RDF-based language that is currently being designed within the W3C/WAI "Evaluation and Repair Tools" working group; see [6]) would have simplified somewhat this step.
- The method cannot be used to analyze other characteristics of the tools that affect their quality of use. It has been defined to focus on more limited properties in order to yield possibly less useful but sounder results that are obtainable through an undemanding evaluation procedure.

An interesting step for refining the method would be to use it, in a group of persons, to run a larger-scale comparison (i.e. with more web sites, a larger sample of issues) perhaps with some additional steps aimed at cross-validating the classification. In such a context it should also be possible to test the *repeatability* of the comparison method, a property that cannot be demonstrated on the single case study reported in this paper.

The method is going to be used by EuroAccessibility [8], a European consortium aimed at supporting and coordinating accessibility methods within European Union member states. In such a context a study will be performed aimed at evaluating repeatability of the method.

Acknowledgments

Many thanks to Jim Thatcher and Daniela Ortner for their detailed reading of a draft of this paper. I'd like to thank also participants of the first face-to-face meeting of EuroAccessibility Task Force 2 held in London, Nov. 2003 for their feedback on the method.

Of course the author is the only one responsible for the content of this paper.

References

1. G. Brajnik. Automatic web usability evaluation: what needs to be done? In *Proc. Human Factors and the Web, 6th Conference*, Austin TX, June 2000. <http://www.dimi.uniud.it/~giorgio/papers/hfweb00.html>.
2. G. Brajnik. Towards valid quality models for websites. In *Proc. Human Factors and the Web, 7th Conference*, Madison, WI, 2001. <http://www.dimi.uniud.it/~giorgio/papers/hfweb01.html>.
3. G. Brajnik. Using automatic tools in accessibility and usability assurance processes. Unpublished note; <http://www.dimi.uniud.it/~giorgio/papers/assurance-proc-note/index.html>, Nov 2002.
4. G. Brajnik. Comparing accessibility evaluation tools: results from a case study. In L. Ardissono and A. Goy, editors, *HCITALY 2003: Simposio su Human-Computer Interaction*, Turin, Italy, Nov 2003. SigCHI-Italy.
5. T. Brink and E. Hofer. Automatically evaluating web usability. CHI 2002 Workshop, April 2002.
6. W. Chisholm and S. Palmer. Evaluation and Report Language (EARL) 1.0. <http://www.w3.org/TR/EARL10>.
7. R. Dougherty and A. Wade. Vischeck. <http://www.vischeck.com/>, Feb 2004.
8. EUROAccessibility. www.euroaccessibility.org. <http://www.euroaccessibility.org>, Nov 2003.
9. N.E. Fenton and S. Lawrence Pfleeger. *Software metrics*. International Thompson Publishing Company, 2nd edition, 1997.
10. R. Gunning. *The techniques of clear writing*. McGraw-Hill, 1968.
11. M. Ivory and M. Hearst. The state of the art in automated usability evaluation of user interfaces. *ACM Computing Surveys*, 4(33):173–197, Dec 2001.
12. M. Ivory, J. Mankoff, and A. Le. Using automated tools to improve web site usage by users with diverse abilities. *IT & Society*, 1(3):195–236, Winter 2003. <http://www.stanford.edu/group/siqss/itandsociety/v01i03/v01i03a11.pdf>.
13. Nielsen Norman Group. Beyond ALT Text: Making the Web Easy to Use for Users with Disabilities. <http://www.nngroup.com/reports/accessibility/>, Oct 2001.
14. M. Paciello. *Web Accessibility for People with Disabilities*. CMP Books, 2000. ISBN: 1929629087.
15. D. Scapin, C. Leulier, J. Vanderdonckt, C. Mariage, C. Bastien, C. Farenc, P. Palanque, and Bastide R. Towards automated testing of web usability guidelines. In *Proc. Human Factors and the Web, 6th Conference*, Austin, TX, June 2000. <http://www.tri.sbc.com/hfweb/scapin/Scapin.html>.
16. J. Slatin and S. Rush. *Maximum Accessibility: Making Your Web Site More Usable for Everyone*. Addison-Wesley, 2003.
17. T. Sullivan and R. Matson. Barriers to use: Usability and content accessibility on the web's most popular sites. In *Proc. 1st ACM Conference on Universal Usability*, 2000.

18. J. Thatcher. Evaluation and repair tools. Used to be posted on <http://www.jimthatcher.com>, June 2002. No more available.
19. Jim Thatcher, Cynthia Waddell, Shawn Henry, Sarah Swierenga, Mark Urban, Michael Burks, Bob Regan, and Paul Bohman. *Constructing Accessible Web Sites*. Glasshouse, 2002.
20. UsableNet Inc. Usablenet technology. http://www.usablenet.com/usablenet_technology/usablenet_technology.html, Nov 2003.
21. UsableNet Inc. LIFT for Dreamweaver — Nielsen Norman Group edition. http://www.usablenet.com/products_services/lfdnng/lfdnng.html, Feb 2004.
22. W3C Web Accessibility Initiative. Evaluation, repair, and transformation tools for web content accessibility. <http://www.w3.org/WAI/ER/existingtools.html>.
23. World Wide Web Consortium – Web Accessibility Initiative. Checklist of checkpoints for web content accessibility guidelines 1.0. <http://www.w3.org/TR/WCAG10/full-checklist.html>, May 1999.
24. World Wide Web Consortium – Web Accessibility Initiative. Web content accessibility guidelines 1.0. <http://www.w3.org/TR/WCAG10>, May 1999.