Improving Browsing Environment Compliance Evaluations for Websites

Cyntrica Eaton
4166 A. V. Williams Building
Department of Computer Science
University of Maryland, College Park, MD 20742
E-mails: ceaton@cs.umd.edu

Abstract

Identifying accessibility issues that can threaten universal website usability is critical for web service and content providers who wish to accommodate the diverse web audience. Detecting page-to-environment incompliance and modifying pages to promote universal accessibility is one important step in improving the process of exploration and navigation in the web user experience. To address this issue, we have designed a system that evaluates the accessibility of a web page in a given browsing environment based on knowledge of the HyperText Markup Language (HTML) tags that comprise the page and knowledge of the tag support provided in respective browsing environments. Given this approach, one of the most important aspects of the system is the comprehensive nature of tag support knowledge. The more support rules known, the more environment-specific bugs the system can accurately identify. In order to optimize knowledge of tag support criteria, we have also incorporated a learning mechanism that can inductively determine HTML tags that are unsupported in a given environment by observing both positive and negative examples of web page appearance and behavior.

1 Introduction

1.1 Motivation

Having long outgrown its novelty [27], the World Wide Web has evolved into an indispensable resource [2] for providing and accessing information and services [7]. Increased reliance on the benefits that stem from a globally interconnected system coupled with the demands and expectations of the growing web community have collectively driven a relevant research effort directed toward improving all aspects of web technology[7]. Addressing challenges that threaten to diminish universal accessibility is one aspect of this multifaceted endeavor as well as the motivation for the work presented here.

While there is a significant research effort directed toward improving accessibility for web constituency with sensory, cognitive, and physical limitations, another important, yet less heavily studied web usability factor is accessibility constraints imposed by end-user browsing environments. The existence and use of diverse, heterogeneous combinations of browser, browser

version, and operating system can cause web page presentation and functionality to vary significantly among users. Web pages that render and function as intended in one environment may be subject to missing page elements, ill-formatted layouts, and erroneous scripts in another [2]. Since the web audience is so diversely equipped, it is impossible to know the combination of web browser, browser version, and platform that will be utilized by all site visitors unless a given website is being designed for a controlled intranet. Consequently, web developers who do not adequately evaluate browsing environment accessibility run the risk of restricting the ability to view and interact with website content to a subset of the web population, ultimately limiting audience reach. The fundamental aim of this research is to provide web developers with an effective way of identifying browsing environment-specific accessibility issues that could hinder users from fully exploring and interacting with the information and services featured on a given website. Effective accessibility evaluation techniques are no less than a necessity for web content providers interested in maximizing audience reach by identifying the browsing environment influenced accessibility issues that could hinder some users from fully exploring their site.

1.2 Current Approaches

One of the most widely used browsing environment accessibility evaluations amongst web developers is dynamic, or execution-based. In this approach, web browsers are the primary evaluation tool, and developers essentially load web pages in a variety of browsing environments and observe subsequent presentation and functionality. Although this evaluation strategy permits a first-hand account of existing accessibility issues, limitations on time and other available resources could severely limit the depth of the website tested and the breadth of browsing environments explored. In most execution-based evaluations, a subset of target environments are identified and reserved as testing platforms leaving many other prospective client environments untested. As a result, the range of confidence in the accessibility of the website is substantially restricted and users with unanticipated browser, browser version, and platform combinations are subject to substandard presentation and functionality.

An alternative, highly effective quality assurance evaluation that can be used to assess web page viability across a series of client environments is the code review, a static analysis technique aimed at identifying fragments of code consistently associated with faulty behavior. In the domain of web applications, the source of these faulty code fragments, or bug patterns, is quite straightforward; browsing environment obstacles arise when unrecognized, or unsupported, HyperText Markup Language (HTML) tags are encountered in document source code. More specifically, a page that renders correctly in one browsing environment may be significantly defective in another based on the relative support of the tags contained in the document source code. Consequently, HTML tags are important accessibility predictors when support for a given tag is known to be nonexistent or insufficient; evaluating the compliance of a web page within an environment can be reduced to identifying browsing environment-specific bug patterns. Since this type of static analysis does not require execution and can essentially use HTML source code as the basis for ensuring universal access, the depth and breadth of evaluation is expected to improve considerably in comparison to execution-based approaches.

1.3 Contributions

Given the relatively efficient nature of accessibility evaluation based on tag support criteria, we have developed a tool that will estimate website-to-browsing environment compliance based on recognition of incompliant, or unsupported, tags within web page source code. This tool essentially automates the code review process for web pages and highlights accessibility threats based on knowledge of the tags that comprise the web pages and the support provided for those tags in various browsing environments. In direct relation with the evaluation scheme, one of the most important aspects of this tool is that it incorporates comprehensive knowledge of tag support rules so that the possibility of false negatives, or erroneous labeling of an incompliant page as accessible in a given environment, is greatly reduced. In other words, conducting code reviews with faulty knowledge can severely inhibit accurate compliance analysis, causing a web page to appear as if it will render properly in an incompliant environment. In order to address this issue, we have incorporated an inductive learning mechanism that can estimate the compliance of HTML tags based on observations of positive (accessible) and negative (inaccessible) examples of web page presentation and functionality. Note, positive examples of web pages can be accessed and utilized as intended by the developer. Negative examples of web pages, on the other hand, are faulty in a given environment and render improperly for users. The underlying theory of the learning technique is that observation of HTML tags that are positively correlated with negative examples can provide insight into the root causes of accessibility issues.

In this paper, we discuss the technique that we have developed for evaluating browsing environment accessibility based on tag support criteria as well as the learning mechanism we have incorporated for updating knowledge of tag compliance. Our approach is expected to be effective because it can identify accessibility barriers across a wide variety of browsing environments with less time and resources than execution-based techniques and it can derive rules about tag compliance based on examples of web pages launched in the field. The paper is arranged as follows: Section 2 outlines the basis of the approach in addition to background definitions that support the underlying theory for the tool. Section 3 provides an overview of the tool and its components. Section 4 covers an initial feasibility study. Section 5 provides insight into related work. Finally, Section 5 outlines future work and concludes.

2 Background

Subjective presentation of web pages across various browsing environments is largely a residual effect of the *Browser Wars*, a period of strong competition among browser vendors at the dawn of the web. Although the web was conceived and initially implemented as a platform neutral, device independent means of accessing information [29] and HTML was originally intended to be a simple language for describing information layouts, several browser developers incorporated proprietary tags that were exclusively supported by their products. While development and introduction of new HTML tags was initially expected to encourage a positive drive toward improved capabilities and more control over page layout, the frustration, loss of productivity, and in some cases, loss of revenue that has resulted from inconsistent support of tags across browsing environments has undoubtedly had a negative impact on the web user experience. Recognizing a need to correct this problem, the World Wide Web Consortium (W3C) [32] set out to define a set of standard tags that all browsers should support. Theoretically, if all browser vendors adhered to the standards, web users would be able to view a web page using

any given browser and gain access to consistent presentation and functionality of the corresponding page. While some browsers claim to be standards compliant, there is evidence that most truly are not in the sense that tags deemed standard by the W3C remain unsupported or are supported improperly [6]. As a result, using the W3C validators is not an adequate measure of cross-browser and, more specifically, cross-browsing environment compliance.

As stated before, a better measure for determining the cross-browser compliance of a web page, at finest granularity, and a website on the whole is to identify incompliant tags within document source code and to report predicted accessibility threats to the developer. Since unsupported tags are most likely to be correlated with rendered errors, they can be considered *bug patterns*. As a result, to recast the meaning of an earlier statement, an effective code review strategy for accessibility evaluation can essentially compare HTML source code of a document with the bug patterns associated with a given client environment. The strength of such a code review, however, would be heavily reliant on the comprehensiveness, or completeness of the set of bug patterns. To further illustrate these ideas, we present the following definitions:

Definition 1: Environment Specific Bug Patterns
Let E denote a browsing environment defined by the triplet $<B, V, O>$ where B is the browser, V is the browser version, and O is the operating system. Consider T, the space of all possible HTML document source tags:

$$\{\{\forall e_j \in E \; \exists I = \{i_1, i_2, \ldots, i_{|I|}\} s.t. (I \subseteq T) \wedge (unsupported(I, e_j)\}\} \tag{1}$$

That is, each browsing environment supports only a subset of the overall tag space T. All other tags are unrecognized or incompliant in the associated environment. Consequently, tags in I could be considered bug patterns for web pages rendered in environment e_j and cross browser accessibility of a website can be evaluated in a code review by detecting the presence of tags $i_n \in I$.

Example 1:
The tag <marquee>, though a part of the comprehensive tag set T and supported by Internet Explorer, is unsupported in browsing environments that feature Netscape. Consequently, <marquee> would be an element of I for environments E where B in the corresponding triplet is Netscape.

The tool we have developed employs this evaluation strategy and compares the HTML tags appearing in the source document of a web page to an inventory of bug patterns associated with various browsing environments. While other tools, such as Doctor HTML [10] and Bobby [4] incorporate similar assessment techniques, we have integrated a mechanism that takes into account the need for a complete and accurate definition of I for each client environment. Consider the following:

Definition 2: Bug Pattern Knowledge Completeness
The accuracy of a code review based on the set of tags in I is largely dependent upon the accuracy and completeness of the description of I. If for instance:

$$\{\exists t_i \in T s.t. (t_i \notin I) \wedge unsupported(t_i, e_j)\} \tag{2}$$

performance of static analysis that does not include t_j as a bug pattern will be compromised.

Example 2:

Consider <blink>, a tag that is unsupported in browsing environments featuring Internet Explorer. If compliance evaluation was executed for a web page that incorporated the <blink> tag yet the tag was not listed in I as a bug pattern, the accuracy of the resulting report would be compromised, and developers would be subject to latent failures and false confidence in universal accessibility.

To address the problem of defining a comprehensive representation of I, we have incorporated a learning mechanism that can extract this knowledge from observations of pages that have positive and negative presentation and functionality in a given environment. Although tag support knowledge is available, obtaining a comprehensive list of the issues that exist within several different types of environments is relatively difficult. Most information stores of tag support knowledge only concentrate on Netscape or Internet Explorer compliance since they are considered the more popular browsers. Given the fact that there is greater diversity in user browsing environments, it is important that we get a better sampling of the tag support rules in more environments to accurately evaluate browsing environment accessibility.

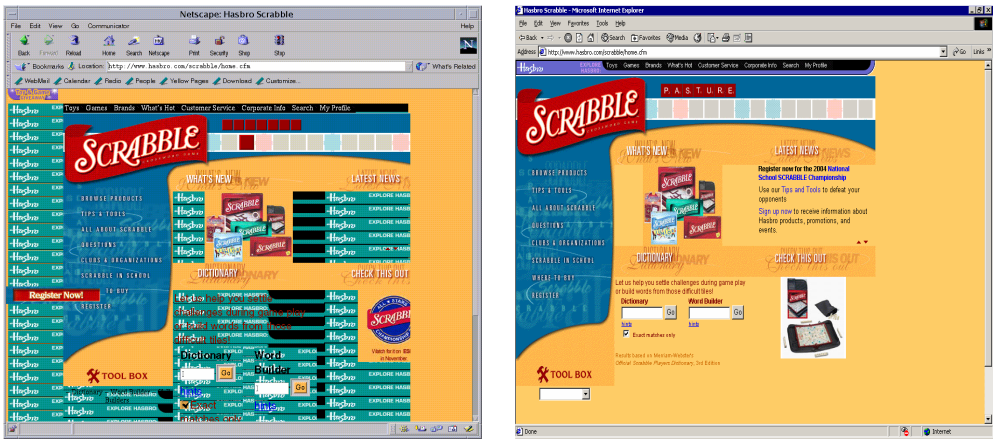
3 Tool Overview

As shown in Figure 1, users with different browser, browser version, and operating system combinations can experience a dramatic imbalance in web page appearance and performance. In this case, the Netscape 4.8 XP Professional environment is unable to process the HTML directive, , and the image repeats in the rendered page as a result. The tool we have developed would examine the source code for this web page, compare it against an inventory of tags known to be subjectively supported in a variety of browsing environments, such as the example provided above, and return a report indicating all detected accessibility threats.

To support the effort to effectively acknowledge and identify existing accessibility threats, there are essentially three main tasks of the tool we developed. They include:

- acquiring comprehensive, complete knowledge of tag support criteria within and across browsing environments,
- utilizing tag support knowledge to evaluate accessibility threats for web pages within a website,
- and producing an accessibility evaluation report outlining the existing accessibility threats and their corresponding environments.

These tasks are carried out, respectively, by the *Tag Support Knowledge Base,* the *Compliance Evaluator,* and the *Accessibility Report* Generator. An overview of how these components work together in the system is shown in Figure 2; a discussion of each component follows in subsequent sections.



Netscape 4.8 XP Professional Internet Explorer 6.0 XP Professional

Figure 1: An example of the significant impact browsing environments can have on accessibility.

3.1 Tag Support Knowledge Base

Adequate population of the Tag Support Knowledge Base is one of the most important aspects of the tool since attempting to detect accessibility threats with faulty knowledge can severely inhibit an accurate report of web page compliance; consequently, developing an adequate knowledge base is imperative to identifying all accessibility threats that exist within a website. To support tag criteria knowledge acquisition, we have incorporated two distinct methods:

- web developers can provide rules (Section 3.1.1),
- rules can be inductively learned based on observations of positive and negative examples of web pages (Section 3.1.2).

3.1.1 Manual Acquisition of Tag Support

Manual acquisition of tag support criteria involves accepting tag support rules directly from web developers. Since documentation on tag support criteria exists, it is possible for web developers to gain access to these rules from various sources and manually enter them into the system. In addition, however, this feature allows developers to specify, or design, arbitrary tag rules to examine customized tag-related issues of interest. This essentially empowers users to perform custom evaluations beyond the scope of browsing environment accessibility making the tool more flexible and subsequently allowing more customized analysis.

While manual acquisition of tag support criteria can be very useful, making web developers solely responsible for providing all necessary support criteria is potentially problematic. In particular, a bottleneck could develop since developers would be responsible for both gathering information and making it available to the tool; also, the resulting accessibility analysis could be compromised since the rules provided might be less than comprehensive. For instance, if a support rule was not explicitly provided to the tool, it would not be applied during subsequent analysis; this would undoubtedly have a negative impact on analysis accuracy. To help combat

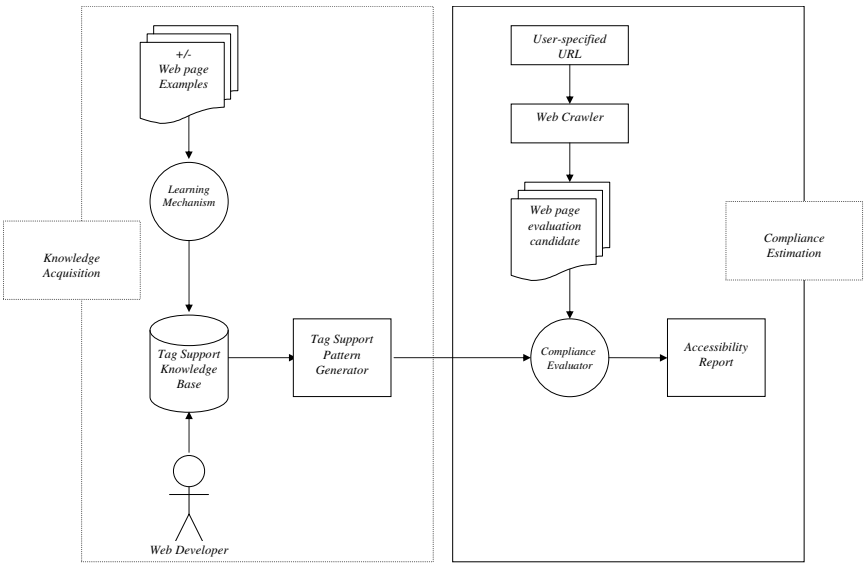


Figure 2: System Overview

these issues, we have incorporated an automatic acquisition method as well.

3.1.2 Automatic Acquisition of Tag Support

Recognizing the benefit of having an automated method for gathering tag support in improving the accuracy of rules retrieved and reducing the burden of the web developer in providing support criteria, we have implemented a learning algorithm; the goal of the algorithm will be to determine the likelihood that an HTML tag is incompliant, or unsupported, given the magnitude of positive correlation with faulty web pages. In terms of the definitions provided earlier, the effort to maintain an accurate, comprehensive knowledge base of unsupported tags, I, associated with a given environment, e_j, can greatly benefit from machine learning methods by refining the definition of I based on observation of pages that worked properly in an associated environment (positive examples) and those that were faulty (negative examples).

As stated before, the major drawback to execution-based accessibility evaluations is the fact that web developers generally lack the time and environment access necessary to perform a thorough assessment. The fact that execution-based techniques feature a first hand account of accessibility issues is actually quite attractive, however, and could provide more insight into existing problems. In designing the learning mechanism, we have established an infrastructure in which average users can submit the URL of a web page they deem to be inaccessible or improperly rendered in their given environment over a normal browsing session; this essentially allows first-hand accounts of faults encountered to be factored into the analysis. In the next section, we offer insight into how the results of user experiences can be used to support detection of accessibility threats in subsequent code reviews.

Learning Technique: In the inductive, or learning by example, methodology presented here, web pages are the raw material for training. The HTML tags that structure the web page and the manually defined classification of the web page as either a positive or negative example provides a statistical basis for determining the influence a given tag has on web page accessibility. To evaluate the association of a given tag with inaccessibility in our current tool, the χ^2

[33] value of the tag will be evaluated. In short, χ^2 uses observance of positive and negative examples to estimate the association of an element to one category or another. In our tool, the χ^2 statistic measures the lack of independence of a tag, t, and a category, c, and evaluates to 0 if the term is independent. The equation is provided below:

$$\chi^2(t,c) = \frac{N \times (AD - CB)^2}{(A+C) \times (B+D) \times (A+B) \times (C+D)} \tag{3}$$

Here, A is the number of positive examples that contain a tag, t; B is the number of negative examples that contain t; C is the number of positive examples that do not contain t, and D is the number of negative examples that do not include the given tag, t.

Web Page Collector: To briefly take a step back, let's consider how the system will acquire the raw material for induction. A web page collector will provide the front end of the learning component, allowing users to submit URLs of both positive and negative examples of web pages in associated environments. Next, the collector will retrieve the corresponding web page and update the values of A, B, C, and D for tags found in the source code; this process will help to establish the likelihood that a given tag is associated with diminished accessibility in an associated browsing environment.

Inference Mechanism: To facilitate the learning process, the inference mechanism incorporates a structure that maintains the χ^2 value for each tag. In particular, each environment is assigned an $n \times 2$ table in which the first column lists the n tags found in examples and the second column of the table hold the χ^2 value for the corresponding tags.

To illustrate the learning strategy and use of the inference structure, consider the following results obtained after observing positive/negative instances and calculating χ^2 for each tag recovered:

Tag	χ^2
`HTML`	0
`Java 1.3`	1.5
`Java 1.2`	2.0
`Java 1.1`	0.33
`Table`	5.0
`Bold`	1.3

Given the data presented and the criteria stated before, the first conclusion that can be drawn is that the `HTML` tag is independent of whether the page is classified as working or faulty. As a result, the inference mechanism can delete this particular tag from the inference table and place it on a list of accepted tags resulting in the following:

Tag	χ^2
`Java 1.3`	1.5
`Java 1.2`	2.0
`Java 1.1`	0.33
`Table`	5.0
`Bold`	1.3

Supported Tags	HTML

Next, the inference mechanism can observe that, in the particular set of examples submitted, the `Java 1.1` tag never occurs in a negative web page although it has appeared in several positive documents provided in the database. The same holds for the `Bold` tag as well. Consequently, those tags can be removed form the inference table and added to the list of accepted tags:

Tag	χ^2
`Java 1.3`	1.5
`Java 1.2`	2.0
`Table`	5.0

Supported Tags	HTML	Java 1.1	Bold

Given this latest development, it can be assumed that `Java 1.3`, `Java 1.2`, and `Table` are possibly faulty tags, and the inference mechanism can use the χ^2 value to estimate the likelihood that the associated tag is associated with faulty web pages. More specifically, when reporting to the result generator, the inference mechanism can incorporate a threshold to determine χ^2 values that correspond with highly probable faulty tags.

3.2 Compliance Evaluator

Once rules have been entered, the system will be primed to evaluate web page accessibility. To begin evaluation, web pages are retrieved from a user-specified root and a web crawler sends the retrieved HTML code to the Compliance Evaluator for analysis. The Compliance Evaluator is essentially responsible for using knowledge of incompliant tags to estimate the prospective accessibility of a given web page within a respective environment. More specifically, using regularized versions of bug patterns stored in the Tag Support Knowledge Base, the Compliance Evaluator accepts a series of web pages from the web crawler, applies the tag-based regular expressions to them, and indicates matches to the report generator.

3.3 Accessibility Report Generator

The accessibility report generator provides tool users with an overview of the pages traversed from a root, most likely a home page, in a hierarchical listing of pages encountered in addition to, and most importantly, an outline of the pages that are expected to fail in a corresponding environment. The hierarchy of the site, as outlined in the tool, provides users with a general idea of the shortest path from the root URL to the page of interest following a series of links. The tool also provides indicators of unreachable, or broken, links as well. An example of this is shown in Figure 3.

Note, in Figure 3, the pane of the interface labeled (3) displays an overview of the pages that are suspected to be faulty in the associated environment. To back track, the area labeled (1) allows users to specify the root URL from which to retrieve subsequent pages, the button labeled (2) allows the user to import a file of customized tag rules and finally, the pane labeled (4)

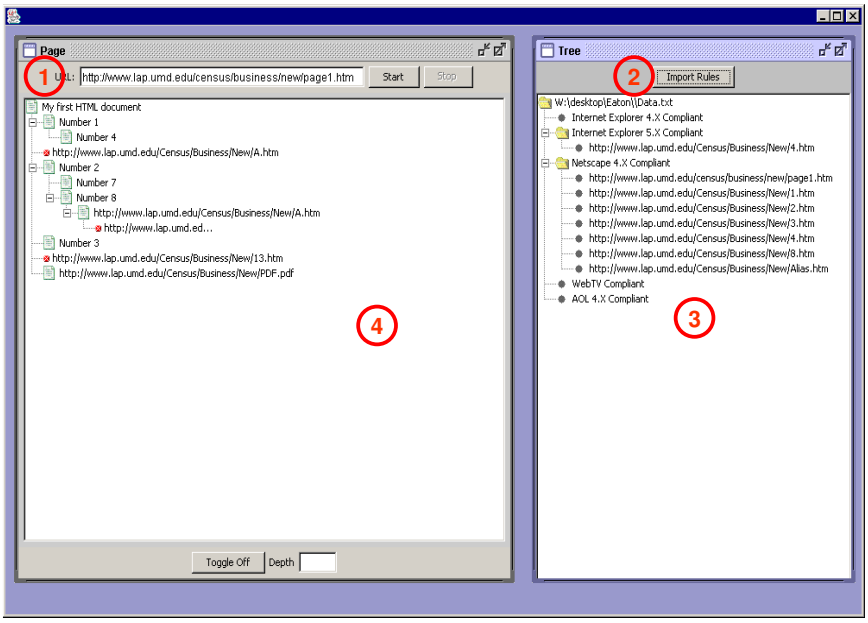


Figure 3: System Interface

provides an overview of the site where the root node corresponds with the root URL provided by the user and all succeeding nodes represent reachable web pages.

4 Feasibility Study

One of the major assertions of our work is that web developers can determine the browsing environment profiles that will not process their web pages accurately based on the tags that structure the page and the support provided for each tag. To evaluate the ability of our tool to do this, we manually provided the tool with a list of tags that were not supported in various environments and applied it to a mock website in order to determine how well the tool would perform. To prime the knowledge base with environment-specific support rules, we consulted several sources on tag support provided in various environments. From the data we attained (a sample of which is shown in Figure 4(a)) we were able to derive an input file of browsing environment bug patterns (a sample of which is shown in Figure 4(b)).

```
Rule Set:  Incompliant Netscape 4.0 HTML
                    <script language = “javascript1.3”>
                    <marquee>
Rule Set:  Incompliant Internet Explorer 5.0 HTML
                    <blink>
                    <body marginwidth=“*”>
                    ……
```

(a)

```
Rule Set:  Incompliant Netscape 4.0 HTML
                    Tag: script
                    Attribute: language
                    Value: javascript1.3
                    Tag: marquee
Rule Set:  Incompliant Internet Explorer 5.0 HTML
                    ……
```

(b)

Figure 4: An example of tag support rules (a) and how developers represent them during manual acquisition (b).

To begin browsing environment accessibility analysis, we provided the homepage URL of the web site as the root in region (1) of Figure 3. Next, we imported an input file similar to Figure 4(b). We were able to observe the arrangement of the website in pane (4) of Figure 3 and

the accessibility threats detected in pane (3) of Figure 3. As a result of our analysis, we were able to quickly identify web pages with possible accessibility threats, the unsupported tags they contained, and the environments they would be unsupported in. Detecting the presence of unsupported tags and presenting them to a web developer who might have used a What You See Is What You Get (WYSIWYG) development tool is expected to make the process of correcting such issues much easier. As a result, we are highly confident in the viability of our tool. We feel it will be highly beneficial to web developers by alerting them of the issues that must be corrected in order to make the information or content they feature available to all web users regardless of browsing environment.

5 Related Work

The overall endeavor to detect browsing environment accessibility barriers and to ensure comprehensive knowledge of tag support falls under general research endeavors associated with web testing, bug isolation, and bug patterns. While the relation to web testing may be obvious, bug isolation is a related endeavor because the attempt to discover new tag support rules is largely an attempt to isolate faulty tags given observations of source documents. Bug patterns, of course, are related because faulty tags can be recast as bug patterns and used during a code review to detect possible accessibility barriers. In this section, we discuss other work that address these issues.

5.1 Web Testing

5.1.1 Web Accessibility

While ensuring and improving accessibility amongst web constituency has been identified as an important issue, most of the work in this area is directed toward improving accessibility for individuals with physical limitations [1, 15, 30], our work, on the other hand, focuses on the less heavily studied issue of accessibility constraints imposed by end-user browsing environments. Published work specifically targeted at improving accessibility evaluations based on browsing environment related constraints [2, 3] is relatively scarce. In addition, there have been relatively few tools created to detect threats to this specific type of accessibility factor [4, 5, 10].

In terms of published work, Berghel [3] presented an accessibility evaluation framework based on the concept of the test patterns used in earlier days to repair and adjust television sets. In Berghel's approach, the so-called *Web Test Pattern* was comprised of a suite of test web pages, each of which incorporated several HTML tags and descriptions of the impact they would have if rendered correctly. This approach essentially allowed users to test their particular client environments to determine compliance levels, but for web developers interested in establishing compliance across a wide variety of platforms, this type of dynamic testing could be severely inefficient. The Browser Photo [5] tool improves on the idea of the test pattern by automatically loading and testing specific pages in a variety of browsing environments and providing screenshots of the results back to the user. Though Browser Photo [5] does not restrict developers to a predefined list of tags and eliminates the need for users to manually load pages themselves, this approach is weakened by the fact that web activity can only be observed above the fold since scrolling to see the entire screen is not an option in individual screenshots; it also

limits the amount of browsing environments that can be tested to the ones featured within the tool.

Other tools that address this problem [4, 10] employ static analysis to identify environment-specific bugs by observing the source code of HTML documents and detecting tags that are known to be incompliant in browsing environments. Yet, again, these tools constrains users to the browsing environments featured and their evaluations are limited by the comprehensive nature of tag support knowledge. As it stands, our system is expected to improve environment accessibility evaluation for tools like these by observing real-word examples of positive and negative page behavior, determining the actual tag support criteria in corresponding browsing environments, and providing this knowledge to such tool so that support criteria can be updated accordingly.

5.1.2 More General Concerns

Ensuring web page quality has become a significant research goal over the years; in general, there has been a great deal of effort placed in applying traditional quality assurance measures to web-based software from object-oriented solutions [7, 9, 24], to reverse engineering techniques [13, 16, 20]. Yet, there are several differences between traditional software systems and the web environment that complicate the endeavor. For one, although traditional software developers had to be relatively familiar with a language before being confident enough to create and distribute products publicly, a growing number of authoring tools providing WYSIWYG tools allow developers to create web pages without being familiar with HTML[30]. The fact that web designers are not necessarily technology specialists threatens the goal of universal quality assurance largely because successful delivery of operable web pages is dependent upon the developers awareness of the aspects involved [27]. While these endeavors share the spirit of our work, assessing the correctness of web applications before they are released in the field, we are most concerned with the challenges presented when web environments are untested and the corresponding environment does not support page elements.

5.2 Bug Isolation and Bug Patterns

On a more general level, there is a body of work related to our overall goal of providing a practical and effective tool for helping software developers detect errors and identify their root causes. In this section, we discuss work involved with bug isolation and bug pattern generation in more traditional software. In [12], Hangal and Lam present DIDUCE, a tool that isolates the root of errors based on identification of system invariants. The work presented in [19] is very similar to our own because the goal is to recognize bugs in systems based on user experiences with faulty executions. The idea is to gather user execution profiles, identify predicates in the source code and, use logistic regression to determine the statements most strongly correlated with system failure. As noted in [11], one of the major obstacles to finding program errors is simply knowing the correctness rules the system must obey. The technique they present looks for contradiction in code constructs, points out the differences, and allows users an opportunity to determine which of the two is incorrect. Once a contradiction is identified, a template rule, or bug pattern, is devised to identify other code that may be the root of similar errors. In [22], Matsumura et. al use bug code patterns to identify violations of implicit coding rules. In other words, legacy systems usually incorporate a series of undocumented, implicit rules that effect execution; new developers to a team may be unaware of them and violate them during maintenance. The idea is to investigate bug reports, identify the error, and derive a bug

pattern that can be used to identify similar code constructs. In our case, novice developers are analogous to new developers on a team unaware of implicit coding rules, individuals in the field encountering faulty executions essentially submit bug reports, and the learning mechanism we have developed identifies the faulty tags and derive corresponding bug patterns.

6 Future Work and Conclusion

6.1 Future Work

The algorithms presented here represent our initial attempts to effectively identify tag-related hindrances to universal accessibility and to devise a plan for inductively determining the tags that are unsupported or incompliant in associated browsing environments. While we are fairly confident in the design we have incorporated for evaluating compliance for individual web pages, we plan to expand the scope of the learning algorithm with further investigation. More specifically, at the very beginning of this project, our initial approach to this problem was to exploit the model of text classification. Yet, instead of detecting words or features most associated with a category, we wanted to determine the accessibility constraints associated with source HTML tags given the strength of association with pre-categorized web pages. In this sense, our interests align because we want to classify the features, or tags, of web documents based on their appearance in, and subsequent association with, accessible and inaccessible pages.

In identifying other fields from which a strong learning algorithm can be based, one highly analogous field is epidemiological study in which the cause of a disease is identified. One particular class of experiment, the case-control study, is most commonly used to study disease etiology and the premise of this type of investigation is in direct alignment with our method for discovering incompliant tags. In short, case-control studies are retrospective in that researchers start with the knowledge of disease occurrence and work backwards to identify any risk factors which can be associated with the outcome. Risk factors, in this domain, are conditions, events, or characteristics which are associated with an increase in risk for a given disease. Similarly, in our mechanism for learning tag compliance, web pages are expected to be preclassified, meaning the outcome is already known. We essentially want to determine the tags most responsible for the classification by identifying tags that have high association with faulty web pages. Given the high level of similarity to case-control studies, one of our future endeavors is to observe the range of techniques employed in case control studies [14, 18, 23, 26] in addition to specific techniques for determining the most significant features in a learning set [8, 17, 28], alternative techniques for determining causation [21, 31], and methods for handling conflicts among data sets in order to identify other, perhaps more appropriate approaches to this problem.

Also, as noted in [25], while building knowledge bases by mass collaboration can greatly improve the time and effort needed to accumulate information, such an approach brings issues such as quality, consistency, and relevance of submitted information and the scalability of learning algorithms used to surface. In our particular case, since the knowledge derived is heavily dependent on examples, it is important that users categorize pages accurately, that examples provide a relevant bases for inference, and that the algorithm is able to process large numbers of contributions accurately. Ensuring that pages provided to the engine actually represent positive and negative examples as labeled and that the population size observed is large enough to derive accurate generalizations will have a direct impact on the success of the approach. Addressing such issues is expected to be a significant part of a future endeavor to maximize the potential of this system on a whole, and the learning mechanism in particular.

6.2 Conclusion

The motivation of this work is one facet of a general endeavor to support and improve the quality of user experiences on the web. The diversity in browsing environments used to navigate the web present a unique challenge for web content providers to effectively asses and correct threats to universal accessibility. Our contribution to this problem has been to derive a framework that can detect accessibility threats based on known bug patterns and to improve knowledge of tag support rules for browsing environments by inductively learning from working and faulty examples of page behavior or appearance. By continuing to observe alternative etiological measures and persistently updating and refining our tool based on our findings, we believe that we can provide an extremely effective defense against the lost productivity and revenue associated with browser-related accessibility issues.

References

[1] Dorothy Ann Amsler. Establishing standards for usable and accessible user services web sites. In SIGUCCS '03: Proceedings of the 31st Annual ACM SIGUCCS Conference on User Services, pages 63–64, New York, NY, USA, 2003. ACM Press.

[2] Hal Berghel. Using the WWW test pattern to check HTML compliance. Computer, 28(9):63–65, 1995.

[3] Hal Berghel. HTML compliance and the return of the test pattern. Communications of the ACM, 39(2):19–22, 1996.

[4] Bobby. http://www.watchfire.com/products/webxm/bobby.aspx.

[5] Browser photo by NetMechanic. http://www.netmechanic.com/browser-index.htm.

[6] Joe Clark. The glorious Peoples myth of standards compliance. http://www.joeclark.org/glorious.html.

[7] Francesco Coda, Carlo Ghezzi, Giovanni Vigna, and Franca Garzotto. Towards a software engineering approach to web site development. In IWSSD '98: Proceedings of the 9th International Workshop on Software Specification and Design, page 8. IEEE Computer Society, 1998.

[8] David R. Cox and E. Joyce Snell. The choice of variables in observational studies. Applied Statistics, 23(1):51–59, 1974.

[9] Pei Hsia D.C. Kung, Chien-Hung Liu. An object-oriented web test model for testing web applications. In Proceedings. First Asia-Pacific Conference on Quality Software, pages 111–120, 2000.

[10] Doctor HTML. http://www2.imagiware.com/RxHTML/.

[11] Dawson Engler, David Yu Chen, Seth Hallem, Andy Chou, and Benjamin Chelf. Bugs as deviant behavior: a general approach to inferring errors in systems code. In SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles, pages 57–72. ACM Press, 2001.

[12] Sudheendra Hangal and Monica S. Lam. Tracking down software bugs using automatic anomaly detection. In ICSE '02: Proceedings of the 24th International Conference on Software Engineering, pages 291–301. ACM Press, 2002.

[13] Ahmed E. Hassan and Richard C. Holt. Architecture recovery of web applications. In ICSE '02: Proceedings of the 24th International Conference on Software Engineering, pages 349–359. ACM Press, 2002.

[14] John H. Holmes. Discovering risk of disease with a learning classifier system. In International Conference on Genetic Algorithms, pages 426–433. Morgan Kaufmann, 1997.

[15] Leonard R. Kasday. A tool to evaluate universal web accessibility. In CUU '00: Proceedings on the 2000 Conference on Universal Usability, pages 161–162, New York, NY, USA, 2000. ACM Press.

[16] Holger M. Kienle and Hausi A. Mller. Leveraging program analysis for web site reverse engineering. In WSE '01: Proceedings of the 3rd International Workshop on Web Site Evolution (WSE'01), page 117. IEEE Computer Society, 2001.

[17] George V. Lashkia. Learning with relevant features and examples. In International Conference on Pattern Recognition, pages 68–71, 2002.

[18] Susan Lewallen and Paul Courtright. Epidemiology in practice: Case-control studies. Community Eye Health Journal, 11(28):57–58, 1998.

[19] Ben Liblit, Alex Aiken, Alice X. Zheng, and Michael I. Jordan. Bug isolation via remote program sampling. In Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation, San Diego, California, June 9–11 2003.

[20] Giuseppe A. Di Lucca, Massimiliano Di Penta, Giuliano Antoniol, and Gerardo Casazza. An approach for reverse engineering of web-based applications. In WCRE '01: Proceedings of the Eighth Working Conference on Reverse Engineering (WCRE'01), pages 231–240. IEEE Computer Society, 2001.

[21] Iain Martel. Probabilistic empiricism: In defence of a Reichenbachian theory of causation and the direction of time. Thesis-University of Colorado, 2000.

[22] Tomoko Matsumura, Akito Monden, and Ken ichi Matsumoto. A method for detecting faulty code violating implicit coding rules. In IWPSE '02: Proceedings of the International Workshop on Principles of Software Evolution, pages 15–21. ACM Press, 2002.

[23] Dirk Pfeiffer and Roger S. Morris. Comparison of four multivariate techniques for causal analysis of epidemiological field studies. In Proceedings of the 7th International Symposium on Veterinary Epidemiology and Economics, pages 165–170, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[24] Filippo Ricca and Paolo Tonella. Analysis and testing of web applications. In ICSE '01: Proceedings of the 23rd International Conference on Software Engineering, pages 25–34. IEEE Computer Society, 2001.

[25] Matthew Richardson and Pedro Domingos. Building large knowledge bases by mass collaboration. In K-CAP '03: Proceedings of the International Conference on Knowledge Capture, pages 129–137. ACM Press, 2003.

[26] Shaun R. Seaman and Sylvia Richardson. Bayesian analysis of case-control studies with categorical covariates. Biometrika, 88(4):1073–1088, 2001.

[27] Brian Sierkowski. Achieving web accessibility. In Proceedings of the ACM SIGUCS Conference on User Services, pages 288–291, 2002.

[28] Keith W. Smillie. Regression analysis: Theory and computation. In Proceedings of the Eigth International Conference on APL, pages 401–407, 1976.

[29] Terry Sullivan and Rebecca Matson. Barriers to use: usability and content accessibility on the web's most popular sites. In CUU '00: Proceedings on the 2000 conference on Universal Usability, pages 139–144, New York, NY, USA, 2000. ACM Press.

[30] C. A. Velasco and T. Verelest. Raising awareness among designers of accessibility issues. In ACM SIGRAPH Computers and the Physically Handicapped, pages 8–3, 2001.

[31] Peter Vineis. Causality in epidemiology. Soz Praventiv Med, 48(2):80–87, 2003.

[32] W3C. http://www.w3.org/.

[33] Yiming Yang and Jan O. Pedersen. A comparative study on feature selection in text categorization. In ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning, pages 412–420, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.