A Cross-browser Web Application Testing Tool[1]

Shauvik Roy Choudhary, Husayn Versee, Alessandro Orso
Georgia Institute of Technology
shauvik@cc.gatech.edu, hversee3@gatech.edu, orso@cc.gatech.edu

***Abstract*—Web applications have gained increased popularity in the past decade due to the ubiquity of the web browser across platforms. With the rapid evolution of web technologies, the complexity of web applications has also grown, making maintenance tasks harder. In particular, maintaining cross-browser compliance is a challenging task for web developers, as they must test their application on a variety of browsers and platforms. Existing tools provide some support for this kind of test, but developers are still required to identify and fix cross-browser issues mainly through manual inspection. Our WEBDIFF tool addresses the limitations of existing tools by (1) automatically comparing the structural and visual characteristics of web pages when they are rendered in different browsers, and (2) reporting potential differences to developers. When used on nine real web pages, WEBDIFF automatically identified 121 issues, out of which 100 were actual problems. In this demo, we will present WEBDIFF, its underlying technology, and several examples of its use on real applications.**

I. INTRODUCTION

Web applications follow a traditional client-server computing model: server-side components receive requests from a client and generate and return responses to such requests. These responses typically consist of HTML documents that contain data, CSS style rules, and client-side scripting components in the form of JavaScript (standardized as ECMAScript) or VBScript. Over the past decade, web applications have evolved from having extremely simple and thin clients to having increasingly rich and complex client-side components.

With this increased complexity of web applications, the maintenance effort required by developers has also increased proportionally. In particular, because the web applications can be run on a variety of web browsers and platforms on the client side, cross-browser compatibility issues are prevalent in such applications. (Current browser statistics report usage of seven popular web browsers across different platforms [2].) *Cross-browser compatibility issues* range from simple cosmetic problems in the user interface to critical functionality failures. According to the latest statistics of the Mozilla's Broken Website Reporter [3], users have reported cross-browser issues on 1,767,900 websites on 463,314 hosts. These figures include 12,072 web sites reported in the past week and 1,568 web sites reported just in the last 24 hours. Because cross-browser issues are observed on the client side (*i.e.,* in the browser) and directly affect the users, there is an increased interest in identifying such issues during in-house testing, before the software is released [4]. In addition, organizations are trying to limit the number of officially supported browsers, which can also negatively affect the user experience [5], [6].

[1]This demo illustrates the implementation of a technique presented in a paper accepted for publication at ICSM 2010 [1].

A. State of the Art

Currently, for detecting cross-browser issues in a given web page, developers must render the page in different browsers and manually inspect the appearance and behavior of the page. Commercial tools, such as Microsoft Expression Web [7] and Adobe's Browser Lab [8], can assist manual inspection by presenting a side-by-side differential rendering of the web application in two browsers. However, these tools still require the developer to spend a considerable amount of effort in manually identifying, understanding, and fixing the actual cross-browser issues. Moreover, some functionality related issues are difficult to spot by manual inspection, as they may not result in readily observable differences.

There are only a few research tools that target cross-browser issues. The compliance assessment technique presented by Eaton and Memon [9] requires developers to manually provide examples of correct and faulty (*e.g.,* containing layout issues) pages, which are then used to compute the probability of a new HTML tag to be faulty. Although, this technique can be useful for simple cases, it requires a considerable amount of manual effort for tagging html pages. Moreover, it does not consider client-side script and CSS style components. More recently, Tamm developed a research tool that leverages both DOM (Document Object Model — http://www.w3.org/DOM) and visual information for finding layout bugs in a particular browser [10]. Tamm's tool is focused mainly on the text portion of a page and identifies web page elements by hiding and showing different elements at different times. In our prior experience with a similar approach, we have discovered that this approach is typically very expensive, as it must render a page a potentially large number of times. Moreover, the approach cannot detect many relevant issues, as it focuses mostly on text elements. The VIPS algorithm [11] infers the hierarchy of a web page from the page's visual layout, rather than from the DOM, by segmenting its screenshot into visual blocks. A limitation of that approach is that it assumes that the page follows a specific layout and groups regions of the page accordingly. However, modern web pages have often complex and creative layouts, and that technique is unlikely to work in these cases. Moreover, disregarding the structural data in the DOM prevents from reporting the specific location in the HTML document that is responsible for an issue, which makes it more difficult for developers to fix the issue.

26th IEEE International Conference on Software Maintenance in Timişoara, Romania
978-1-4244-8628-1/10/$26.00 ©2010 IEEE

B. Proposed Solution

To address the limitations of the existing techniques, we developed WEBDIFF, a technique and tool for (1) detecting cross-browser issues automatically and (2) reporting the locations of such issues in the corresponding web page to help developers fix them. WEBDIFF is based on the concept of differential testing [12]. It loads a web page in different environments and compares the behaviors of the page in such environments; intuitively, a difference in behavior indicates a potential problem.

Our tool can be used in isolation or in combination with existing tools for functional testing, such as Selenium [13], and existing test-input generation techniques for web applications (*e.g.,* [14]–[18]). These tools can be used to generate and run inputs for the web applications under test, while WEBDIFF can compare the web pages generated using such inputs on multiple browsers and platforms.

To assess the effectiveness of WEBDIFF, we performed an empirical evaluation in which we used our tool on nine real web pages. Our results are promising: WEBDIFF automatically identified 121 issues, out of which 100 were actual problems. In this tool demonstration, we will present WEBDIFF, its underlying technology, and several examples of its use on real applications. In the rest of the paper, we describe in greater detail our technique, results, and proposed demonstration.

II. OUR TECHNIQUE

As discussed above, our technique finds dissimilarities between the corresponding elements of a web page rendered in different browsers, one of which is considered to be a "reference browser". Having a reference browser is important to keep the number of comparisons low. Moreover, it mimics a normal scenario where developers focus on making sure that the web application behaves well in their browser of choice and then check whether it behaves consistently in other browsers. More precisely, the technique operates as follows. Given a web page URL, it first opens the page in all considered browsers. Second, it extracts from each browser the DOM information and a screenshot of the rendered web page. Third, it identifies variable elements on the reference browser's DOM and eliminates them from further consideration. Fourth, it matches the nodes in the DOM tree hierarchy amongst different browsers. Finally, for each pair of matched nodes, it compares the attributes of the two nodes and the regions of the screenshots corresponding to the nodes to find inconsistencies. We explain each of these steps concisely in the following sections. More details on the technique can be found in [1].

A. Data Collection

WEBDIFF launches the web page under test in all browsers considered and adjusts the browsers' window size such that they are all equal. The browser size is obtained inside the web application by a simple script and is communicated to the tool that resizes the browser accordingly. More details about the engineering of this step can be found in Section IV. After the browsers are ready, WEBDIFF extracts the DOM information of the web page and captures a screenshot for each browser window. Specifically, the DOM information captured by WEBDIFF for every DOM node consists of the following properties:

- *tagname*: Name of the tag associated with the DOM element.
- *id*: Unique identifier of the DOM node, if defined.
- *xpath*: X-Path of the node in the DOM structure.
- *coord*: Absolute screen position of the DOM element.
- *clickable*: True if the DOM element has a click handler.
- *visible*: True if the DOM element is visible.
- *zindex*: DOM element's screen stack order, if defined.
- *hash*: Checksum of the node's textual content, if any.

WEBDIFF captures the screenshot of the web page by taking a snapshot of the browser window and extracting just the content in the viewport (*i.e.,* the section of the browser where the web page is rendered).

B. Detection of Variable Elements

Web Pages often have generated elements that differ across executions, such as advertisements and statistics about the page. Such variable elements must be ignored during comparison, as they are highly likely to result in false positives. WEBDIFF detects variable elements by loading the web page in the reference browser twice and comparing the DOM and screenshot information obtained in the two cases. The intuition is that the variable elements will likely vary in subsequent requests. All DOM nodes that reveal either a structural or a visual difference in this analysis are marked as variable and ignored in the subsequent steps.

C. Cross-browser Comparison — Structural Analysis

The goal of this phase is to match the DOM nodes obtained from different browsers for the same page. WEBDIFF traverses the DOM trees for two browsers in parallel while searching for the best matching nodes. While analyzing a pair of nodes, the technique computes a *match index*, a number between zero and one that represents how similar the two nodes are. The match index is computed using the DOM properties that are recorded in the previous step (see Section II-A). More precisely, two nodes with the same id are marked as a perfect match, and so are two nodes with the same xpath. If none of these conditions is satisfied the match index for the two nodes is proportional to the similarity of the xpaths for the nodes and the number of other properties that have identical values. When all possible node pairs have been considered, the matching node for a given node n is the node m such that there is no other pair of nodes that (1) contains n and (2) has a match index greater than the match index for the pair (n, m). At the end of this step, each node of the reference browser's DOM is mapped to a node in the DOMs of each other browser considered.

D. Cross-browser Comparison — Visual Analysis

The DOM does not have information on how web page elements exactly appear on screen. Therefore, our technique also performs a visual analysis of the page. In this step, WEBDIFF leverages the structural information computed in the previous

step to (1) identify corresponding elements in the browser screenshots and (2) perform a graphical matching of such elements. As a preliminary step of the graphical matching, WEBDIFF grays out the areas corresponding to variable nodes in the captured screenshots, so as to eliminate possible false positives caused by them. It then checks for four classes of issues: positional, size, visibility, and appearance differences. Positional differences are found by checking the parent (or container) nodes of matching HTML elements and the relative position of the elements with respect to such container nodes. Visibility and size differences are identified by comparing the properties of two matching nodes. Finally, WEBDIFF identifies appearance differences by graphically comparing the regions of a web page that correspond to matching HTML elements. The graphical comparison uses a histogram based technique that can compute the distance between two images [19]. If such distance is larger than a given threshold, the two elements are reported as different in the two browsers considered.

III. USAGE SCENARIOS

Table I lists some scenarios in which the WEBDIFF tool can be useful. In the first scenario, the tool assists the developers in supporting a new web browser. The second scenario describes the use of the tool for regression testing. In the final scenario, WEBDIFF allows developers to reproduce an error that was reported to them, thereby allowing them to investigate the issue for resolving it. In general, WEBDIFF can be used in a variety of scenarios for supporting software development and maintenance tasks.

TABLE I
SCENARIOS OF USE FOR WEBDIFF.

Steps	Expected Output
A new version of a browser is released.	
Add the new browser version to WEBDIFF and run it on the web application.	List of issues in the web pages of the web application related to the new browser version are reported.
A new version of the application is released.	
Exercise WEBDIFF on the new version of the application on all the considered browsers.	List of issues introduced in the new version are reported to the developers.
A cross-browser issue needs to be reproduced.	
Run WEBDIFF on the affected web page of the application on the specific browsers.	The relevant issue gets identified by WEBDIFF, allowing the developer to investigate it.

IV. THE TOOL

A. Architecture

Figure 1 shows a high level architecture of the tool. As shown in the figure, the tool operates in a distributed manner. The main controller runs in the host machine, and the web browsers are installed in a virtual machine connected to it. This setup allows the host to connect to multiple virtual machines, which in turn allows browsers on multiple platforms to be used for testing. Different conflicting versions of the same browser can also be used with such a setup. For instance, currently three different versions of Internet Explorer are popularly used for web browsing and should therefore be used for compatibility testing of web applications. The virtual machine contains several browsers, controlled by the *Data Collection Engine*. The *Data Collection Engine* consists of three main components: a *GUI Automation* component, which automates the tasks of launching browsers, opening web pages, and performing actions on the web page; a *DOM Extraction* component, which is a JavaScript tool to extract the tree based DOM of the web page; and a *Screen Capturing* component, which captures and saves the screenshot of a web page. The host machine consists of the WEBDIFF *Controller*, which is responsible for starting the *Web Proxy* and then remotely starting the capture process on all virtual machines. The *Web Proxy* is necessary to overcome the browser's cross domain restrictions. The *DOM Extraction* script submits the data back to the location where the web page originated from. The *Web Proxy* identifies the HTTP request as a data-capturing request and redirects it to a local server side script that saves the data in the *Data Store* on the host machine. The *Web Proxy* also passes values between the JavaScript *DOM Extraction* script inside the web page and the *GUI Automation* tool. Similar to the data-capturing requests, the *Web Proxy* handles the communication messages in a special manner. These messages are essentially set and get requests to the web service, such that one of them can retrieve the data set by the other. Specifically, the *GUI Automation* tool reads the `scrollWidth` and `scrollHeight` of the web page and decides how much more the web page needs to be resized.

After the DOM and visual information have been collected from the target web page, the *Analysis Engine* reads this information and performs the steps explained in Section II: variable element identification and cross-browser comparison.

B. Technologies Used

The components of WEBDIFF use a varied set of platforms and libraries. Communication between the machines is performed using the Software Testing and Automation Framework (STAF),[2] which is a P2P based cross-platform automation platform. The security settings in STAF are configured to allow the host to start the remote *Data Collection Engine* process on each virtual machine. The *GUI Automation* component in the *Data Collection Engine* process uses the `win32api`[3] to access and perform activities on the web browser window inside the Microsoft Windows virtual machine. The *Screen Capturing* tool uses the `win32api` to obtain the browser window handle and the `python image`

[2]http://staf.sourceforge.net
[3]http://sourceforge.net/projects/pywin32/

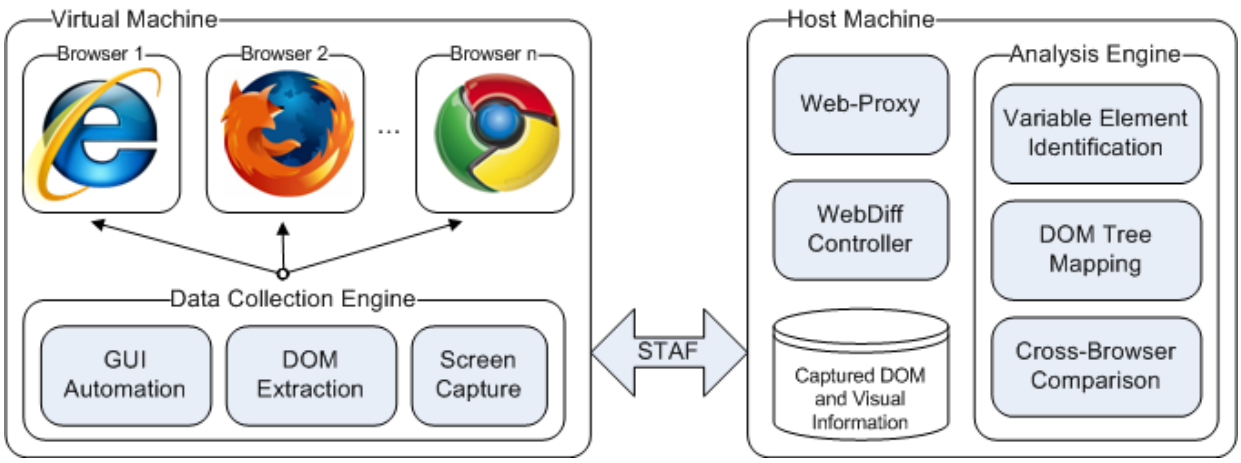
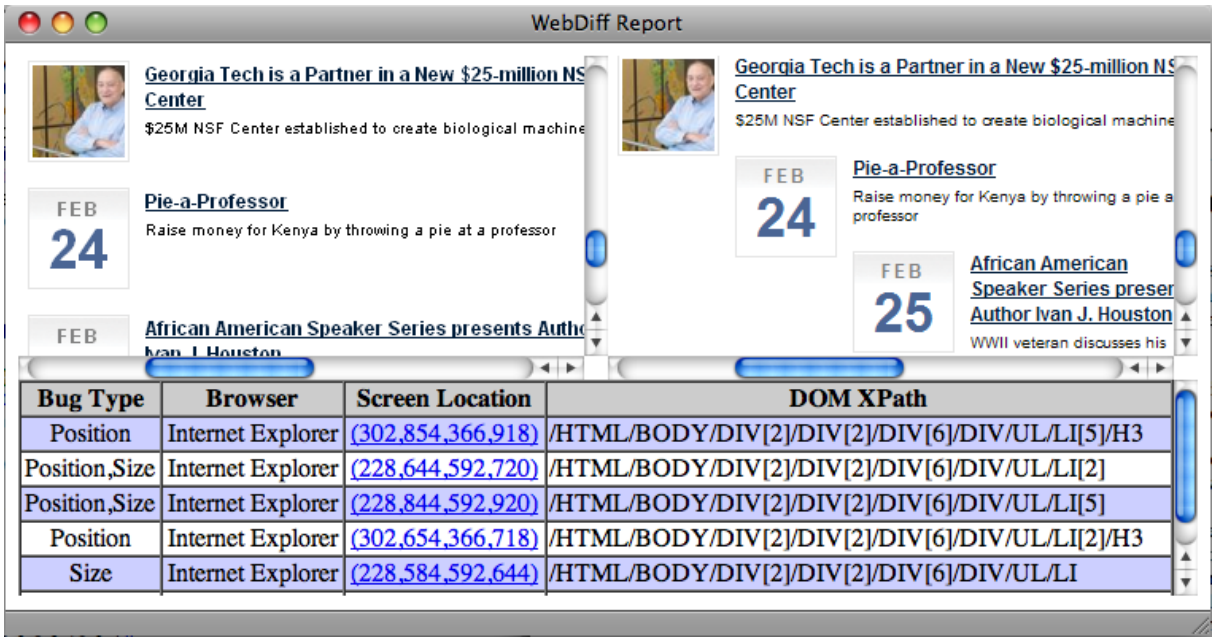


Fig. 1. Architecture of WEBDIFF.



Bug Type	Browser	Screen Location	DOM XPath
Position	Internet Explorer	(302,854,366,918)	/HTML/BODY/DIV[2]/DIV[2]/DIV[6]/DIV/UL/LI[5]/H3
Position,Size	Internet Explorer	(228,644,592,720)	/HTML/BODY/DIV[2]/DIV[2]/DIV[6]/DIV/UL/LI[2]
Position,Size	Internet Explorer	(228,844,592,920)	/HTML/BODY/DIV[2]/DIV[2]/DIV[6]/DIV/UL/LI[5]
Position	Internet Explorer	(302,654,366,718)	/HTML/BODY/DIV[2]/DIV[2]/DIV[6]/DIV/UL/LI[2]/H3
Size	Internet Explorer	(228,584,592,644)	/HTML/BODY/DIV[2]/DIV[2]/DIV[6]/DIV/UL/LI

Fig. 2. Report generated by WEBDIFF.

`library`[4] to obtain and save the screen captures of a web page. On the host machine, the WEBDIFF *Controller* and *Web Proxy* are written in the python programming language, and the latter uses the `twisted` [5] networking library for making network connections. Inside the *Analysis Engine*, the Variable Element Identification and DOM tree mapping components are written in Java. The Cross-Browser Comparison component is partly written in Java (for the part comparing matched DOM locations) and partly in C using the Open Computer Vision library[6] (for the part performing image processing for screenshot comparison).

[4]http://www.pythonware.com/library/
[5]http://twistedmatrix.com
[6]http://opencv.willowgarage.com

C. Reporting of Issues Found

Figure 2 shows a sample report generated by WEBDIFF. The report is an HTML page that consists of two major sections: visual section and data. The top visual section shows a screenshot of the web page in the reference browser, along with the same in another browser. This visual section can be used by developers to quickly inspect the screenshots side by side. The lower data section contains the list of issues found in tabular format. The list contains the type of issue, the browser on which it was found, and the corresponding screen and DOM locations of the element that manifested the issue. As the figure shows, the WEBDIFF report is intuitive and can be useful to find and fix cross-browser issues.

Fig. 3. Georgia Tech web site.

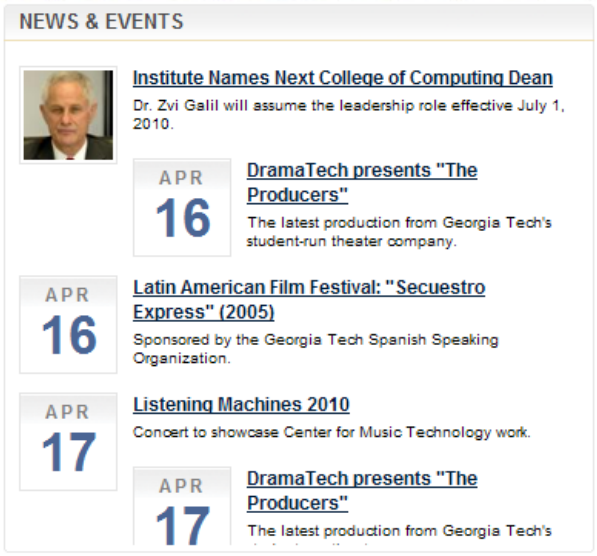


Fig. 4. Issues with Georgia Tech's web site: shifted elements in IE.

V. Evaluation

For evaluating our tool, we selected three widely used web browsers: Mozilla Firefox[7] (Version 3.6), Google Chrome[8] (Version 4.1), and Internet Explorer[9] (Version 8.0). We installed the browsers in a Microsoft Windows XP virtual machine. For the evaluation, we selected nine web pages as subjects. The first subject is the front page of our institutional web site, http://www.gatech.edu, shown in Figure 3. As it can be seen in the figure, the web site uses a professional template and makes use of a variety of HTML features. This page was selected due to our prior knowledge of its cross-browser issues, so that we could check whether WebDiff identified them. Some of these issues are clearly visible in Figure 4: in the web page, some elements appear shifted to the right when compared to the page shown in Figure 3.

The remaining eight pages that we used for the study were chosen randomly. To perform a fair sampling of subjects, we picked them using a random link generator service provided by Yahoo! (http://random.yahoo.com/bin/ryl). The complete list of web pages considered is shown in Table II. As the table shows, the list includes a wide range of web applications, ranging from static informational web pages to commercial web sites.

The results of our study are shown in Table III. The numbers listed in the table are the total number of issues found while comparing the web pages rendered in Firefox (considered as the reference browser) with pages rendered in Chrome and Internet Explorer. The table reports, for each subject, the different types of cross-browser issues identified by WebDiff that were manually confirmed as true positives: positional differences (*Pos.*), changes in size (*Size*), visual differences (*Vis.*), and general appearance differences (*Gen.*). In addition, the table shows the total number of true positives (*Tot.*) and the total number of false positive (*FP*) reported.

As the results in Table III show, WebDiff was able to automatically discover and report a large number of cross-browser issues: 121 issues of different types overall. In particular, WebDiff was able to identify the known issues in the Georgia Tech web site. WebDiff reported a total of 21 false positives for the nine subjects considered in the study. This corresponds to a 17% false positive ratio, which can be considered acceptable given that WebDiff is the first tool for automatically detecting such problems. Moreover, in six out of nine subjects (a majority), the false positive rate is even lower. Manual analysis of the results showed that the false positives reported by the tool were due to (1) minor differences in some elements containing text and background images that are mostly unnoticeable for the human eye and (2) presence of some variable elements that WebDiff failed to identify. After studying the issues, we believe that they can be eliminated by careful engineering of the tool.

We also performed a manual checking of the web pages considered in the study, and the checking did not reveal any additional cross-browser issue that our tool had not revealed. In other words, to the best of our knowledge, the tool generated no false negatives.

One last point that is worth discussing is the cost of our approach. The analysis time for each of the nine web pages was less than five minutes. Because WebDiff can be run overnight, this time is definitely acceptable. We therefore did not perform any more detailed study of the performance of WebDiff.

[7] http://www.mozilla.com/firefox/
[8] http://www.google.com/chrome
[9] http://www.microsoft.com/windows/internet-explorer/

TABLE II
SUBJECTS USED FOR THE STUDY.

Subject	URL	Type
GATECH	http://www.gatech.edu	university
BECKER	http://www.beckerelectric.com	company
CHESNUT	http://www.chestnutridgecabin.com	lodge
CRSTI	http://www.crsti.org	hospital
DUICTRL	http://www.duicentral.com	laywer
JTWED	http://www.jtweddings.com	photography
ORTHO	http://www.otorohanga.co.nz	informational
PROTOOLS	http://www.protoolsexpress.com	company
SPEED	http://www.speedsound.com	e-commerce

TABLE III
NUMBER OF CROSS-BROWSER ISSUES IDENTIFIED.

	# Faults identified					
Subject	*Pos.*	*Size*	*Vis.*	*Gen.*	*Tot.*	*FP*
GATECH	2	3	0	1	6	0
BECKER	2	12	0	2	16	1
CHESNUT	8	4	0	2	14	2
CRSTI	4	4	0	2	9	0
DUICTRL	9	8	0	2	19	4
JTWED	3	9	0	1	14	0
ORTHO	0	0	0	2	2	2
PROTOOLS	4	5	0	2	11	9
SPEED	23	5	0	2	30	3
Total	55	50	0	16	121	21

VI. CONCLUSION

Cross-browser issues are increasingly prevalent, and achieving consistent behavior across all major browsers has been a serious concern for web application developers. Current techniques for detecting and fixing such issues are immature and require a considerable amount of manual work from the developers. Our tool, WEBDIFF, addresses the limitations of existing techniques by automatically identifying cross-browser issues. To do so, WEBDIFF compares web pages by leveraging both their structural (web page's DOM) and visual (web page's snapshots) characteristics. We evaluated WEBDIFF on nine real world web applications, and the results of the evaluation show that the tool is effective in identifying cross-browser issues while keeping the false positive rate low.

This demo paper presented our tool, its underlying technique, and some of its implementation details. The paper also presented some possible scenarios of usage for the tool and our initial empirical evaluation. In future work, we will investigate ways to further lower the number of false positives (*e.g.,* by better detecting variable elements and improving the accuracy of our graphical matching) and to improve the tool based on feedback from our users. We will also perform more experiments on additional subjects to confirm our initial results. In the longer term, we will investigate the use of our approach to evaluate behavioral equivalence of web applications that can run not only in different browsers, but also on completely different platforms (*e.g.,* web applications running on both desktop and mobile platforms).

TOOL AVAILABILITY

The tool has been released under MIT license, for public download and use. More information about the tool can be found at the following location: http://www.cc.gatech.edu/~shauvik/webdiff.php

ACKNOWLEDGMENTS

This work was supported in part by the NSF awards CCF-0916605 and CCF-0725202 to Georgia Tech.

REFERENCES

[1] S. Roy Choudhary and A. Orso, "Webdiff: Automated identification of cross-browser issues in web applications," in *ICSM '10: Proceedings of the International Conference on Software Maintenance*. IEEE, September 2010.

[2] W3Schools.com, "Browser statistics month by month," http://www.w3schools.com/browsers/browsers_stats.asp, May 2010.

[3] Mozilla, "Firefox broken website reporter," http://reporter.mozilla.org/app/stats/, July 2010.

[4] Cambridge Network, "Estate agents must update web browser compatibility ahead of microsoft announcement," http://www.cambridgenetwork.co.uk/news/article/default.aspx?objid=69332, March 2010.

[5] The Korea Times, "Korea sticking to aging browser," http://www.koreatimes.co.kr/www/news/biz/2010/02/123_61463.html, February 2010.

[6] N. Gohring (IDG News), "Google to end support for ie6," http://www.pcworld.com/article/188190/google_to_end_support_for_ie6.html, January 2010.

[7] Microsoft, "Expression web," http://www.microsoft.com/expression/products/Web_Overview.aspx, May 2010.

[8] Adobe, "Browser lab," https://browserlab.adobe.com/, May 2010.

[9] C. Eaton and A. M. Memon, "An empirical approach to evaluating web application compliance across diverse client platform configurations," *Int. J. Web Eng. Technol.*, vol. 3, no. 3, pp. 227–253, 2007.

[10] M. Tamm, "Fighting layout bugs," http://code.google.com/p/fighting-layout-bugs/, October 2009.

[11] D. Cai, S. Yu, J.-R. Wen, and W.-Y. Ma, "Vips: a vision-based page segmentation algorithm," Microsoft Research, Tech. Rep., November 2003.

[12] W. M. McKeeman, "Differential testing for software," *Digital Technical Journal*, vol. 10(1), pp. 100–107, 1998.

[13] OpenQA, "Selenium web application testing system," http://seleniumhq.org/, May 2010.

[14] D. Roest, A. Mesbah, and A. v. Deursen, "Regression testing ajax applications: Coping with dynamism," in *Software Testing, Verification and Validation (ICST), 2010 Third International Conference on*, 6-10 2010, pp. 127 –136.

[15] S. Artzi, A. Kiezun, J. Dolby, F. Tip, D. Dig, A. Paradkar, and M. D. Ernst, "Finding bugs in dynamic web applications," in *ISSTA '08: Proceedings of the 2008 international symposium on Software testing and analysis*, 2008, pp. 261–272.

[16] F. Ricca and P. Tonella, "Analysis and testing of web applications," in *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2001, pp. 25–34.

[17] X. Jia and H. Liu, "Rigorous and automatic testing of web applications," in *In 6th IASTED International Conference on Software Engineering and Applications (SEA 2002*, 2002, pp. 280–285.

[18] W. G. J. Halfond and A. Orso, "Improving test case generation for web applications using automated interface discovery," in *ESEC-FSE '07: Proceedings of the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM, 2007, pp. 145–154.

[19] Y. Rubner, C. Tomasi, and L. J. Guibas, "The earth mover's distance as a metric for image retrieval," *International Journal of Computer Vision*, vol. 40, pp. 99–121, 2000.