WEBVIZOR: A Visualization Tool for Applying Automated Oracles and Analyzing Test Results of Web Applications

Sara Sprenkle[†], Holly Esquivel[§] [*], Barbara Hazelwood[¶][*], Lori Pollock[‡]
[†] Washington & Lee University, sprenkles@wlu.edu
[‡] University of Delaware, pollock@cis.udel.edu
[§] University of Wisconsin-Madison, esquivel@cs.wisc.edu
[¶]Cisco Systems, bahazelw@cisco.com

Abstract

Web applications are used extensively for a variety of critical purposes and, therefore, must be reliable. Since Web applications often contain large amounts of code and frequently undergo maintenance, testers need automated tools to execute large numbers of test cases to determine if an application is behaving correctly. Evaluating the voluminous output—typically Web pages full of content—is tedious and error-prone. To ease the difficulty, testers can apply automated oracles, which have tradeoffs in false positives and false negatives. In this paper, we present the design, implementation, and evaluation of WEBVIZOR, *a tool that aids testers by applying a set of oracles to the output from test cases and highlighting the symptoms of possible faults. Using* WEBVIZOR *a tester can compare the test results from several executions of a test case and can more easily determine if a test case exposes a fault.*

1 Introduction

Businesses, governments, and consumers depend on the reliable execution of Web applications for a variety of purposes, such as online searching, shopping, banking, and social networking, and are also increasingly being used for more traditional software purposes [18]. Web applications are frequently updated to improve functionality, reliability, usability, and appearance [15]. Since Web applications evolve and must be reliable, testers often perform maintenance and regression testing. In regression testing, a tester executes a test suite on a version of the application that is known to behave correctly and then executes the same test suite on a modified version of the application, such as one that uses an updated library or alternative data store. The tester will then compare the output from executing the test suite on each version of the application to ensure that the modification did not result in incorrect behavior. While there has been success in automating test case generation and execution [8, 12, 19, 20, 21, 10] and load testing [7, 10] for Web applications, a remaining challenge is to develop *automated support for verifying that executed test cases produced correct results*.

This challenge is not unique to Web applications: one of a tester's greatest challenges is interpreting the voluminous test results from executing test suites [9]. With respect to Web applications, testers need to manage many types of output generated by Web applications, including HTML pages, email messages, data and server state. A tester may choose to focus on a Web application's HTML output, which is reasonable because users see the HTML page and errors in other output may propagate to the HTML output [16]. Still, given the amount of information crammed into an HTML page [13], a tester examines a lot of output—the rendered Web page and possibly its dense HTML source.

A tester can write an oracle manually, using a testing framework such as Cactus [1], WebTest [4], HtmlUnit [11], or Selenium [19], but writing the oracles manually is tedious and error-prone, resulting in missed failures, i.e.,*false negatives*, or reporting real-time, dynamic content as failures, i.e., *false positives*. Reporting false positives leads to wasted developer effort tracking down nonexistent bugs. Instead of manually writing an oracle, a tester could apply a `diff` on the expected and actual HTML pages [5], but then a tester could again mistakenly report false positives because of dynamic content. In previous work, Sprenkle et al. developed a suite of 22 *automated* oracle comparators specialized for Web applications' HTML output [22]. We found that the comparators showed tradeoffs in false positives and false negatives, that there was no single ideal comparator for all applications, and that combinations of oracles sometimes achieved better effectiveness.

[*]Author contributed while at the University of Delaware with support from the CRA-W Distributed Mentor Project.

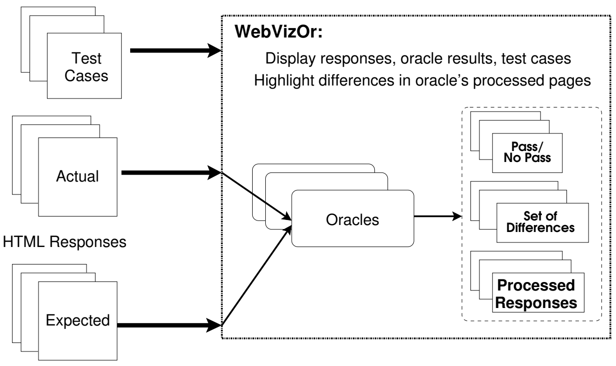


Figure 1. Web Application Testing Process with WEBVIZOR

If a tester chooses a more conservative automated comparator, i.e., one that reports more false positives but fewer false negatives, she may want to verify manually that a test case indicates a fault before wasting developer effort. Without appropriate visualization tools, verification may not be cheaper than manually creating oracles.

To address these challenges, we have developed WEBVIZOR (WEB application failure detection VIsualiZation with ORacles), an open-source tool that aids testers in the analysis of test results for Web applications. By applying customized, automated oracle comparators and integrating test information (e.g., test cases, responses, and oracle results), WEBVIZOR helps the tester identify failures and their corresponding test cases. WEBVIZOR can also be used to evaluate oracle comparators and direct the development of new oracles. Specifically, WEBVIZOR provides functionality to

- apply automated, customized oracle comparators to test results and plug in new oracles,
- visually compare actual and expected test results as well as output from a set of oracle comparators, which provide different views of the Web page, to ease determining if a test case passes,
- view test results in the context of its test case, and
- navigate through test results.

In this paper, we describe WEBVIZOR's features, design, implementation, and typical use and present our experience using WEBVIZOR in a large empirical study.

2 Background

Figure 1 illustrates WEBVIZOR in the context of testing Web applications. We view a test case for a Web application as a sequence of HTTP requests sent to the Web application under test and the test results as the responses—typically HTML documents—generated by executing the requests. The HTML responses comprise the *actual results* from testing. The *expected results* are provided from executing the suite on a previous, working version of the Web application.

A *test oracle* is a mechanism to evaluate the actual results of a test case as "pass" or "no pass". An oracle produces an expected result for an input and uses a *comparator* to verify the actual results against the expected results [3]. Oracles have traditionally been difficult and expensive to develop in software testing [2, 3, 17, 23]. The primary challenge to developing oracles for Web applications is their low *observability*, i.e., while some application output goes to the user in the form of generated HTML pages, other application behaviors, such as generated email messages, internal server state, data state, and calls to Web services, are not as easy to monitor.

WEBVIZOR's integrated oracles focus on validating HTML output—a subset of the application's output [23]. Existing Web application testing approaches [1, 4, 6, 8, 12, 19] have also employed test oracles that use the HTML responses as the output of an executed test case. As shown in Figure 1, WEBVIZOR's oracles perform customized processing on the actual and expected responses and report *pass* or *no pass* based on the differences between the processed HTML responses [22].

The next section focuses on how WEBVIZOR applies the oracle comparators and displays the test input and output to users in an easy-to-use interface.

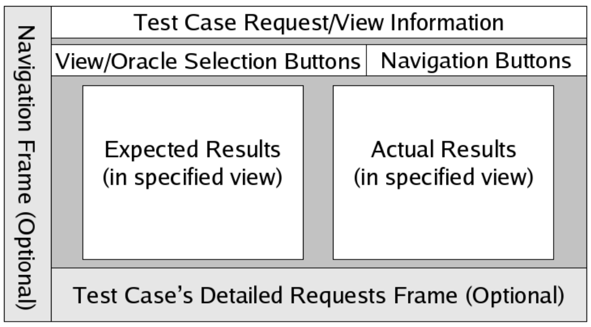
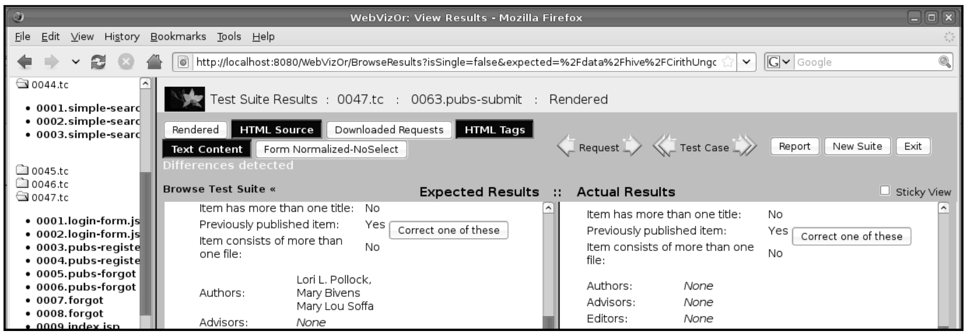

Navigation Frame (Optional)
Test Case Request/View Information
View/Oracle Selection Buttons
Navigation Buttons
Expected Results (in specified view)
Actual Results (in specified view)
Test Case's Detailed Requests Frame (Optional)

Figure 2. WEBVIZOR GUI: Feature Overview

3 Features

WEBVIZOR provides functionality to apply automated, customized oracle comparators to test results; visualize actual and expected test results for easy comparison; visualize output from a set of oracle comparators, which also provide different views of the test results; navigate through test results for failure detection and analysis; and view the test results in context with the test case. This section describes WEBVIZOR's features, as shown in their


WebVizOr: View Results - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/WebVizOr/BrowseResults?isSingle=false&expected=%2Fdata%2Fhive%2FCirithUngc
Google
Rendered
HTML Source
Downloaded Requests
HTML Tags
Text Content
Form Normalized-NoSelect
Request
Test Case
Report
New Suite
Exit
Browse Test Suite «
Sticky View

0044.tc
- 0001.simple-searc
- 0002.simple-searc
- 0003.simple-searc

0045.tc
0046.tc
0047.tc
- 0001.login-form.js
- 0002.login-form.js
- 0003.pubs-registe
- 0004.pubs-registe
- 0005.pubs-forgot
- 0006.pubs-forgot
- 0007.forgot
- 0008.forgot
- 0009.index.jsp

Test Suite Results : 0047.tc : 0063.pubs-submit : Rendered

Differences detected

Expected Results :: Actual Results

Expected Results		Actual Results	
Item has more than one title:	No	Item has more than one title:	No
Previously published item:	Yes	Previously published item:	Yes
Item consists of more than one file:	No	Item consists of more than one file:	No
Authors:	Lori L. Pollock, Mary Bivens, Mary Lou Soffa	Authors:	None
Advisors:	None	Advisors:	None
		Editors:	None

Correct one of these

Figure 3. WEBVIZOR in Rendered view. The buttons of oracles reporting failures are highlighted.

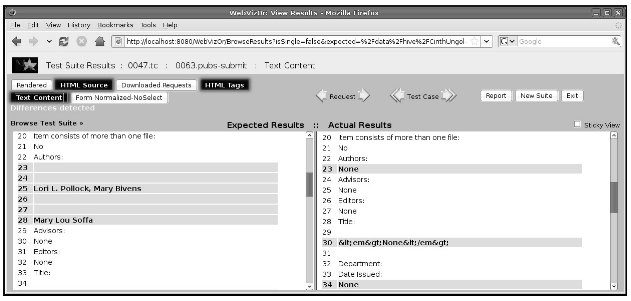
typical GUI layout in Figure 2.

View and Visually Compare Test Results. WEBVIZOR presents the test results via a *single view* or *comparison view*. The single view mode allows the tester to quickly inspect the HTML document in either its rendered (i.e., browser-displayed) or HTML source code form in a single frame. The comparison view allows for easy comparison of the actual and expected responses to the same request in adjacent frames in one of several views: rendered, HTML source (i.e., the **Document** comparator [22]), or as the processed HTML results from one of a set of oracle comparators, such as those presented in [22].

View Oracle Comparator Results. Viewing rendered and source-code forms of an HTML response can be an inefficient way to analyze possible manifestations of faults in actual responses; thus, an essential part of our tool is the ability to view pairs of HTML responses after various oracles analyzed them. When a user navigates to a pair of actual and expected responses, WEBVIZOR automatically executes each test oracle comparator. An oracle's Selection Button (shown near the top of Figure 2) is highlighted if the oracle comparator reports that the actual response contains a failure. By selecting the appropriate oracle button, a user can view the oracle comparator's processed actual and expected HTML responses and the highlighted differences between the responses (if any). A user can configure WEBVIZOR to use their own oracle comparators as well.

By viewing oracle comparator results, a tester can more easily determine if a reported failure is truly a failure. As shown in the screenshots in Figures 3 and 4, a tester can see if multiple oracles agree in reporting the failure and can examine an oracle's reported differences in the oracle's processed responses. For these screenshots, the differences indicate a failure because the authors' names are missing in the actual response.

Navigate through Test Results. Efficiently navigating


WebVizOr: View Results - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/WebVizOr/BrowseResults?isSingle=false&expected=%2Fdata%2Fhive%2FCirithUngol-
Google
Rendered
HTML Source
Downloaded Requests
HTML Tags
Text Content
Form Normalized-NoSelect
Request
Test Case
Report
New Suite
Exit
Browse Test Suite »
Sticky View

Test Suite Results : 0047.tc : 0063.pubs-submit : Text Content

Differences detected

Expected Results :: Actual Results

```
20 Item consists of more than one file:      20 Item consists of more than one file:
21 No                                        21 No
22 Authors:                                  22 Authors:
23                                           23 None
24                                           24 Advisors:
25 Lori L. Pollock, Mary Bivens              25 None
26                                           26 Editors:
27                                           27 None
28 Mary Lou Soffa                            28 Title:
29 Advisors:                                 29
30 None                                      30 &lt;em&gt;None&lt;/em&gt;
31 Editors:                                  31
32 None                                      32 Department:
33 Title:                                    33 Date Issued:
34                                           34 None
```

Figure 4. WEBVIZOR screenshot of responses shown in Figure 3 in Text Content view. Differences between the processed responses in TextContent view are highlighted.

through test results from large test cases and test suites is crucial to ease the burden of Web application testers. In WEBVIZOR, the tester may choose to iterate through the entire execution of a test suite via navigation buttons, which allow the tester to navigate to a previous or next response as well as jump to the previous or next test case in the suite. The tester's ability to navigate through a suite in WEBVIZOR reduces the time it typically would take for the tester to open all the responses for both the expected and actual results individually and process the results from each of the test oracles. The tester also has the option to navigate to a particular test case in the suite from WEBVIZOR's navigation frame. For large test suites, WEBVIZOR automatically organizes the test cases into a hierarchy of folders in the navigation frame.

View Detailed Test Case Information. To analyze why a Web application generates certain responses, testers may want to view a test case's input, i.e., the original HTTP request and its data. In WEBVIZOR, a tester can choose to view all of the detailed requests that comprise the test case. To further utilize the detailed request information, WEBVIZOR allows the tester to click on any of the displayed detailed requests and view the corresponding response.

Apps	Classes	Methods	Statements	NCLOC
Masplas	9	42	441	999
Book	11	385	5250	7791
CPM	75	172	6966	8947
DSpace	274	1453	27136	49513

Table 1. Subject Application Characteristics

4 Design and Implementation

WEBVIZOR is a Web application that executes entirely on the local machine. We implemented WEBVIZOR as two modules: a user-friendly GUI frontend and a modular backend that provides the functionality and framework to plug in new test oracles. The GUI frontend is built using JavaServer Pages (JSPs) and HTML and communicates with the Java servlet backend. Test oracles are executed from the Java backend and are implemented in Perl or Java. WEBVIZOR executes all oracles on an HTML response automatically when the user navigates to the response. We developed WEBVIZOR on Linux and deployed it on both the Resin and Tomcat Web application servers.

WEBVIZOR has several application configuration files. An oracle configuration file allows the user to select the set of oracles to execute. Currently, four basic test oracles have been integrated into WEBVIZOR [20]. Integrating a new Perl or Java-based oracle that adheres to the interface simply requires modifying WEBVIZOR's oracle configuration file.

The current implementation of WEBVIZOR has some limitations. WEBVIZOR assumes that responses are named lexicographically in request order (e.g., our testing framework names the responses in order 0001.req, 0002.req, ...). Depending on the saved HTML source, WEBVIZOR may not exactly duplicate what a user sees because it may not reference the displayed page's images or style sheets. For example, if the source uses relative (rather than absolute) links to external resources, WEBVIZOR does not store those resources locally and therefore can not display them. Even without style sheet information, the tester is able to effectively analyze possible faults (other than style-related ones) detected via oracle comparator results.

5 Experience Using WEBVIZOR

We have used WEBVIZOR to analyze test results, evaluate oracle comparators, and direct the development of new oracles. The best example of WEBVIZOR's use is our research group's application of WEBVIZOR in a large empirical study of the effectiveness of 22 automated oracle comparators [22]. The empirical study is summarized in this section.

The goal of the empirical study was to evaluate the accuracy of oracle comparators—specifically, their reported failures, false positives, and false negatives. We performed the study on four representative, deployed subject applications: a conference registration system, an online bookstore, a course management system, and a digital publications library. Table 1 contains the subject applications' code characteristics. (characteristics in Table 2) on a clean version of the application to generate expected HTML results. We also seeded faults in each application (listed in Table 3) and executed the test suite once for each fault-seeded-version of the application to generate the actual HTML results. The average size of the HTML responses varied between 2.0 KB and 12.5 KB (Table 2). We then compared the actual and expected responses with each of the 22 automated oracle comparators.

			# HTML Pages		
	Seeded	Exposed	Exposing Each Fault		
Apps	Faults	Faults	Median	Mean	Std Dev
Masplas	28	22	35.5	54.9	55.9
Book	39	36	595	521.2	629.3
CPM	135	96	25	202.6	516
DSpace	50	20	104	282.6	347.4
Total	252	174	N/A	N/A	N/A

Table 3. Seeded and Exposed Faults

To determine an oracle comparator's accuracy, we needed to determine the expected oracle results, i.e., which responses contained manifestations of a fault. We started with the results from the **Document** oracle: the oracle that detects all differences in the HTML responses. We then used WEBVIZOR to compare the responses that **Document** reported as failures and determined if the differences truly indicated a failure. Table 3 shows the failure exposure results we determined using WEBVIZOR. Utilizing WEBVIZOR's comparison view, integrated oracles, and navigation tools, the task of analyzing tens of thousands of responses took a few days rather than the weeks the process would take a human manually.

In addition to evaluating oracles, we used WEBVIZOR to develop new oracles. We analyzed test results from automated oracle comparators presented by Sprenkle et al. [20]. We identified common situations where a particular oracle reported false positives or false negatives, which led us to develop new oracle comparators. Using WEBVIZOR, we could more easily recognize similar oracle behavior for certain types of responses.

6 Related Work

Open-source and commercial HTML Diff Visualization tools are available online, including those surveyed by Rick McGowan [14]. However, these tools have the same disadvantages as our **Document** oracle [22] in that they show every minute difference between responses—including irrelevant differences. WEBVIZOR provides additional functionality customized to Web application testers, such as the

Apps	Test Input # Test Cases	 # Requests	% Stmt Coverage	Test Output # HTML Responses	 Avg HTML Response Size
Masplas	169	1103	90.5%	1103	3.8 KB
Book	125	3564	56.8%	3564	10.7 KB
CPM	890	12352	78.4%	12352	2.0 KB
DSpace	75	3183	51.6%	3023	12.5 KB

Table 2. Test Suites and Responses

integration of several customized oracles.

There are many Web site test tools [10]. Some Web site and Web application testing frameworks build on JUnit, e.g., Cactus [1], WebTest [4], HtmlUnit [11], and HttpUnit [12]. These frameworks include functionality to execute test cases but require a tester to create manual oracles for each test case. Selenium is a capture-replay tool that runs directly in a browser such as Firefox or Internet Explorer and also requires manual creation of oracles [19]. A tester can execute test cases using various tools, but only with WEBVIZOR can a tester apply a set of automated oracle comparators to the responses and compare the actual and expected results. Using Selenium, a tester can visualize the responses from test cases, but Selenium does not allow a user to easily compare and navigate through the expected and actual responses.

7 Conclusions and Future Work

We have developed WEBVIZOR, a new tool for visualizing and comparatively analyzing test results from Web applications. WEBVIZOR is mature: we have already used WEBVIZOR successfully to compare test results from different oracles on deployed Web applications with moderately sized test suites as well as to direct development of new oracles. Our future work includes increasing WEBVIZOR's portability on various platforms and adding features to improve visualization and usability. WEBVIZOR is available from `http://www.cis.udel.edu/~hiper/webvizor`.

References

[1] Apache Cactus. `http://jakarta.apache.org/cactus/`, 2007.

[2] L. Baresi and M. Young. Test oracles. Technical Report CIS-TR01-02, University of Oregon, 2001.

[3] R. Binder. *Testing Object-Oriented Systems*. Addison Wesley, 2000.

[4] Canoo WebTest. `http://webtest.canoo.com/`, 2008.

[5] S. Chawathe and H. Garcia-Molina. Meaningful change detection in structured data. In *International Conference on Management of Data*, May 1997.

[6] G. DiLucca, A. Fasolino, F. Faralli, and U. D. Carlini. Testing web applications. In *International Conference on Software Maintenance*, Oct. 2002.

[7] D. Draheim, J. Grundy, J. Hosking, C. Lutteroth, and G. Weber. Realistic load testing of web applications. In *Conference on Software Maintenance and Reengineering*, 2006.

[8] S. Elbaum, G. Rothermel, S. Karre, and M. Fisher II. Leveraging user session data to support web application testing. *IEEE Transactions on Software Engineering*, 31(3):187–202, May 2005.

[9] D. Hoffman. A taxonomy for test oracles. Quality Week '98, `www.softwarequalitymethods.com/Papers/OracleTax.pdf`, 1998.

[10] R. Hower. Web site test tools and site management tools. `http://www.softwareqatest.com/qatweb1.html`, 2008.

[11] HtmlUnit. `http://htmlunit.sourceforge.net/`, 2008.

[12] HttpUnit. `http://httpunit.sourceforge.net/`, 2008.

[13] S. Krug. *Don't Make Me Think*. New Riders Press, Aug. 2005.

[14] R. McGowan. Uncle Rick's guide to visual HTML diff products. `http://rm-and-jo.laughingsquid.org/Text/Guide-To-Differs.html`, 2003.

[15] J. Offutt. Quality attributes of web software applications. *IEEE Software: Special Issue on Software Engineering of Internet Software*, 19(2):25–32, March/April 2002.

[16] S. Pertet and P. Narsimhan. Causes of failures in web applications. Technical Report CMU-PDL-05-109, Carnegie Mellon University, Dec. 2005.

[17] D. Richardson. TAOS: testing with analysis and oracle support. In *International Symposium on Software Testing and Analysis*, 1994.

[18] Rubicon Consulting. Growth of web applications in the US: Rapid adoption, but only when there's a real benefit. White Paper, `http://rubiconconsulting.com/`, Sept. 2007.

[19] OpenQA: Selenium. `http://www.openqa.org/selenium/`, 2008.

[20] S. Sprenkle, E. Gibson, S. Sampath, and L. Pollock. Automated replay and fault detection for web applications. In *International Conference on Automated Software Engineering*, Nov. 2005.

[21] S. Sprenkle, E. Gibson, S. Sampath, and L. Pollock. A case study of automatically creating test suites from web application field data. In *Workshop on Testing, Analysis, and Verification of Web Services and Applications*, July 2006.

[22] S. Sprenkle, L. Pollock, H. Esquivel, B. Hazelwood, and S. Ecott. Automated oracle comparators for testing web applications. In *International Symposium on Software Reliability Engineering*, November 2007.

[23] E. Weyuker. On testing non-testable programs. *The Computer Journal*, 25(4):465–70, 1982.