An Oracle based on Image Comparison for Regression Testing of Web Applications

Akihiro Hori*, Shingo Takada*, Haruto Tanno† and Morihide Oinuma†
*Dept. of Information and Computer Science, Keio University, Yokohama, Japan
{hori, michigan}@doi.ics.keio.ac.jp
†Software Innovation Center, NTT Corporation, Tokyo, Japan
{tanno.haruto, oinuma.m}@lab.ntt.co.jp

***Abstract*—Much work has been done on automating regression testing for Web applications, but most of them focus on test data generation or test execution. Little work has been done on automatically determining if a test passed or failed; testers would need to visually confirm the result which can be a tedious task. The difficulty is compounded by the fact that parts of a Web page (such as advertisements) may change each time the Web application is executed even though it has no bearing on the Web application function itself. We thus propose a test oracle for automatically determining the result of regression testing a Web application. The key point of our approach is the identification of parts that may change, which we call variable region. We first generate the expected result, by executing the original (pre-modification) Web application multiple times so that variable regions can be identified. Then, after the Web application is modified, regression testing is conducted by comparing the output of the modified Web application against the expected output. An evaluation confirmed the usefulness of our approach.**

Keywords- web application testing; regression test; image comparison

I. INTRODUCTION

The Web application is a popular form of application due to the user basically only needing a Web browser to access it. They, however, tend to be modified frequently for various reasons, including bug fix, enhancements, and security attacks [1][2].

As with any software, each time a Web application is modified, it must go under regression testing, which checks that functions that performed correctly before modification still do. Although much work has been done on automatic regression testing [1][3][4][5][6][7][8], most focus on the automation of test data generation, selection, prioritization, or its execution.

An important part of regression testing is to check the result of test data execution against the expected result. This may be done automatically through means such as DOM tree comparison [8] and XML/HTML output [4]. However, in many cases, this is done manually [9][10], especially for testing that relies on visually checking the screen display (i.e., browser output) [11]. Such manual checking is time consuming as human testers must check each Web page one by one; this is compounded by the fact that checking each Web page needs to take place each time the application has been modified [10].

We thus focus on the automation of checking if the screen display has changed or not. Specifically, we target the development of a *test oracle* for Web applications.

A test oracle can be defined as having two essential parts [12]:

1) oracle information that represents expected output
2) an oracle procedure that compares the oracle information with the actual output

In our approach, we automatically generate the expected output for each test case by executing the test case multiple times and saving the resulting screen display as an image. After the Web application is modified, its screenshot is also saved and compared with the expected output. If the images are the same, the test case is said to have passed, otherwise it has failed.

Although the basic steps are simple, there is one important difference between the output of Web applications and conventional GUI applications. There may be regions within the output of Web applications that may change each time that Web page appears regardless of the actual results of the Web application execution. We call such a region as *variable region*. For example, many Web applications have pages that contain advertisements, which may change each time that page is shown. Such regions must be accounted for; otherwise even if the Web application result is correct, a change in the advertisement being shown will cause the image comparison to fail. Thus an important part of our approach is the elimination of these variable regions.

This paper thus proposes an oracle for the regression testing of Web applications. The main contributions are as follows:

- Automatic identification of variable regions by executing the Web application multiple times before modification.
- Automatic generation of an expected result (baseline image) for a given test case based on the identification of variable regions.
- Automatic "PASS/FAIL" judgement of test case execution by comparing screenshots taking into account variable regions.

The rest of this paper first starts with a discussion of related work. Section 3 then describes our approach. Section 4 evaluates our approach. Section 5 makes concluding remarks.

DOI 10.18293/SEKE2015-017

II. Related Work

Web application testing focusing on the visual aspect of the output includes [11][13][14][15][16].

Stocco, et al. proposed an approach to migrate test suites based on DOM (Document Object Model) [17] to test suites based on images [11]. Although the goal of their work is different from ours, there are aspects similar to our work; specifically, the mapping of DOM elements with visual elements of the Web page.

Choudhary, et al. proposed an approach to detect cross-browser incompatibilities in Web applications [13]. Although the goal of their work is different from ours, they also target eliminating variable regions. However, their approach generates the expected output by executing the pre-modification Web application twice. If the variable regions *always* change, this would be sufficient, but this is not the case. Our approach solves this issue.

Selay, et al. [10] proposed an approach that compares images for regression test. Their approach was able to efficiently detect layout faults while neglecting insignificant variations. However, their "insignificant variation" does not include our variable region. Our approach thus differs from this perspective.

Applitools [15] is a tool that automatically tests Web applications in a variety of environments (e.g., OS, browsers), and saves screenshots. After the tests are executed, the saved images are shown one by one, and testers need to determine "PASS/FAIL" one by one. Our approach goes a step further as the "PASS/FAIL" determination is done automatically.

PhantomCSS [14] is a tool that automatically executes regression testing of Web applications by comparing screenshots of modified Web applications with screenshots of pre-modification Web applications. The goal of their tool is the same as ours, but variable regions can only be eliminated manually. Our approach solves this issue by automatically removing variable regions.

Screenster [16] is a tool that automatically determines "PASS/FAIL" of tests. Developers first record the initial result of executing a Web application. After making changes to the Web application, it is executed and the execution result is compared with the initial result. If the images are the same, the test is determined as pass. Otherwise, differences in the two results are highlighted. Testers then manually check if the differences can be ignored (in which case the test passes) or not (in which case the test fails). Although Screenster can automatically determine "PASS/FAIL", it cannot handle the dynamic parts (variable region), while our approach can.

In sum, although much work has been done on visual testing of Web applications, the determination of "PASS/FAIL" is still done manually. PhantomCSS and Screenster can make an initial determination of "PASS/FAIL", but they cannot automatically handle variable regions. Our proposed approach addresses this issue.

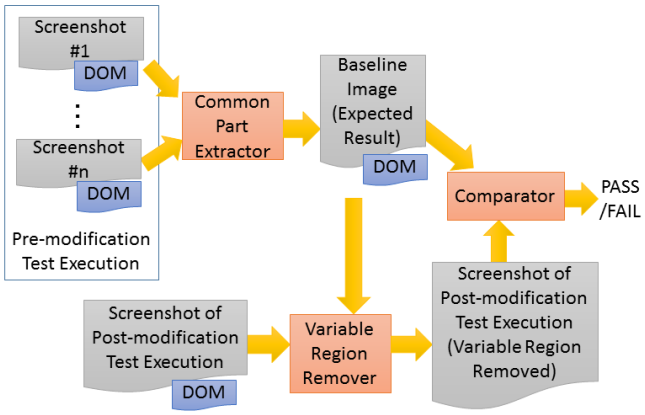

Fig. 1. Overview of proposed approach

III. Image Comparison Based Test Oracle

A. Overview of our approach

Fig. 1 shows the overview of our proposed approach. The basic steps are as follows:

1) Execute a test case multiple times before the program is modified, and produce n screenshots of the resulting Web page. We call this *pre-modification test execution*. The number of times (n times) is set by the tester.
2) Extract the common parts of n screenshots (subsection III-B). This eliminates the variable regions, and the resulting image serves as the expected result for the regression test.
3) Execute the test with the modified program, and produce a screenshot of the resulting Web page.
4) Remove the variable region from the resulting Web page (subsection III-C).
5) Compare the baseline image with the post-modification image (subsection III-D). If the two images match, then the regression test result is PASS. Otherwise, it is a FAIL.

B. Common Part Extraction

The common part extraction first divides each screenshot into several regions based on the DOM tree (Fig. 2). The DOM tree is traversed from its root until an element reaches a specified size, at which point all children of that element is deleted (Fig. 3). Each leaf in the remaining tree is considered as a "region". Note that all n screenshots will have the same DOM tree structure, and thus the same resulting regions.

The element size is set by the tester. If the threshold takes too large a value, a region that contains both variable parts and non-variable parts will be designated as being a variable region. On the other hand, too small a value will lead to many regions which will cause unnecessary computation.

The common part extraction continues by comparing the regions one by one based on the position within the DOM tree. The set of regions that are the same with all n screenshots are

Fig. 2. Separation into region images

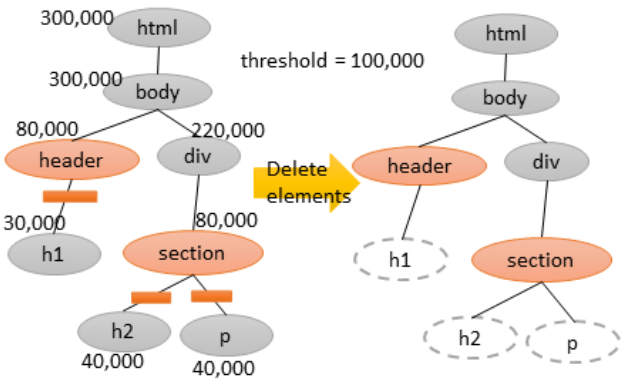

Fig. 3. The search and delete of elements in the DOM tree (Values express the size of the element)

stored in a log file (which we call *pre-modification common part log file*). Each log in this file is a pair of the image of the region and its DOM tree absolute path. This set of regions forms the baseline image, i.e., the expected output. Regions that differ at least once are considered as variable regions.

The comparison between each region, i.e., the comparison between two images, is done by calculating the distance between the two regions, and then checking if the distance is less than a specified threshold. If the distance is less than the threshold, the two regions (i.e., images) are considered to be the same. If it is greater than or equal to the threshold, they are considered to be different.

We adopted the χ^2 histogram distance for the region comparison, which is often used in image processing. It has been found to be accurate and computationally fast [13]. χ^2 histogram distance is computed as follows:

$$\chi^2\left(H_1, H_2\right) = \sum_i \frac{\left(H_1\left(i\right) - H_2\left(i\right)\right)^2}{H_1\left(i\right) + H_2\left(i\right)} \quad (1)$$

H_1 and H_2 are the histograms of the images. These histograms are the distribution of the luminance, i.e., i indicates the luminance value and $H\left(i\right)$ indicates the number of pixels that has luminance value i.

C. Variable Region Removal

The variable region removal process starts by first dividing the screenshot image of the post-modification Web page into regions in the same way as the common part extraction. The DOM tree is then used to remove the regions corresponding to the variable regions. Specifically, the DOM tree absolute path for each region is checked to see if it exists in the *pre-modification common part log file*. If it exists, then the image of that region will be compared with the corresponding baseline image in the next step. Otherwise, that region is considered as a variable region and ignored in the next step.

D. Comparison

This module compares the corresponding regions based on the DOM tree between the baseline image and the post-modification image with the variable regions removed. The algorithm used to compare each region is the same as the one used during the common part extraction (section III-B). If all regions are the same, the regression test has passed. Otherwise, it has failed.

E. Tool Implementation

We implemented our proposed approach on top of a testing tool that we had previously developed [18] [19]. In our previous tool, users first define what we call a *base scenario*, which corresponds to a set of steps that end-users take when they use a given Web page "normally". Test cases are then generated by using the steps in the base scenario, and searching a knowledge-base to find similar steps. Found steps will include information such as constraints on user input and previously used test data that are used for the generated test cases.

The generated test cases are then executed using Selenium Web Driver [20], and the screen display is saved. Our previous work was able to automate many steps that are required for testing Web applications, but it could not automatically judge if a test case execution passed or failed. Our proposed approach makes this possible.

We list below values that the tester can specify:

- REPEAT indicates the value of n, i.e., the number of times the test is executed pre-modification. The default value is 3.
- SIZE is the threshold specifying the minimum size of elements in the DOM tree (Fig. 3). The default value is 100,000 pixels.
- DISTANCE is the threshold for determining if two regions are considered to be the same or not. The default value is 10.
- INTERVAL is the wait time between pre-modification tests. This is to account for cases where variable regions may change only after a certain amount of time has passed. The default value is 0.

The default values were specified based on preliminary experiments that we conducted.

IV. Evaluation

We conducted a case study to evaluate our approach, focusing on the following three research questions:

- RQ1: Does our tool judge as correctly as humans?
- RQ2: Does our tool judge more quickly than humans?
- RQ3: Does our tool remove more variable regions when the number of times of the pre-modification test is bigger?

RQ1 checks how accurate our tool is compared to manual human comparison. RQ2 checks the time taken to compare images. Since automation is supposed to help humans, our tool needs to be as accurate (or nearly as accurate) as humans, and it also needs to be faster. RQ3 checks whether the repeated execution of pre-modification test is meaningful.

A. Target Applications

We targeted three demo Web applications: Zen Cart [21], Takai [22], and Welcart [23]. Zen Cart is a shopping cart software. Welcart is also a shopping cart software, but it is a WordPress plugin. Takai is a free Joomla template, that can be used to build Web sites. We chose these Web applications because they are open source, have variable regions, and have sample data available.

We used the provided programs as the original (pre-modification) applications. The modifications were done by changing the programs using the concept of mutation analysis [24]. which basically changes or *mutates* a part of the code. We employed 26 basic mutation operators each of which changes one operator, such as changing $<$ to $>$. Although mutating the program basically means that we are injecting a bug into the program, the goal of this case study is to check if our tool can accurately judge if there has been a change in the output Web page, and *not* to evaluate the test suite. In a true testing environment, there will be additional test cases that check for changes that were made to the program. Thus, some mutated code will result in the output Web page being different from the original output, while others will be the same. Adding new test cases to account for changes made to the program is out of scope of this paper.

Test cases were generated using our previously developed tool [18][19]. Twenty-eight test cases were generated for Zen Cart, five test cases for Takai, and twelve test cases for Welcart.

The case study was conducted using FireFox and specified the following values:

- REPEAT: 6 for RQ1 and RQ2; 1 to 6 for RQ3
- SIZE: 100,000
- DISTANCE: 15
- INTERVAL: 0

TABLE I
Accuracy rate (%)

	Tool	Human
Zen Cart	99.4	99.8
Takai	98.0	90.7
Welcart	100	100
Total	99.4	98.8

B. Manual Judgement

We asked five students to compare pairs of Web page images, and judge if each pair is the same or not. For each pair, one Web page image was the result of executing the original program, while the other was randomly chosen from the mutated results. The variable region was specified for each pair by highlighting the appropriate part of the Web page. In other words, each student subject needed to check the non-variable regions, and did not need to consider the variable region(s) when judging if a pair was the same or not. This is because in an actual testing environment, the tester will know in advance which part(s) of the Web page will change each time it is loaded.

C. Results

1) RQ1: Accuracy: We carefully analyzed both the results of our tool and the results of the humans manually, and checked if the judgement was correct or not.

Table I shows the results. The accuracy rate of our tool was nearly as high as the human result.

Tables II - V show the breakdown of the results. "Tool" denotes the result from our tool, while "Solution" denotes the correct result, which was obtained from careful manual analysis by the authors. "P" means *pass*, and "F" means *fail*. The numbers indicate the number of test data. So, for example, in Table II, our tool correctly identified 607 *pass*es and 228 *fail*s, while incorrectly identifying 5 *fail*s as *pass*es. The only cases of a *pass* incorrectly being identified as a *fail* occurred in the Takai Web application.

TABLE II
ZEN CART

		Solution	
		P	F
Tool	P	607	5
	F	0	228

TABLE III
TAKAI

		Solution	
		P	F
Tool	P	106	0
	F	3	41

TABLE IV
WELCART

		Solution	
		P	F
Tool	P	111	0
	F	0	249

TABLE V
TOTAL

		Solution	
		P	F
Tool	P	824	5
	F	3	518

TABLE VI
EXECUTION TIME (SECONDS)

	Tool	Human
Zen Cart	4.2	48.1
Takai	3.7	64.2
Welcart	2.4	27.7
Total	3.7	44.4

2) RQ2: Execution Time: We measured the execution time for our tool. Specifically, we measured the time taken for variable region removal and comparison (Fig. 1) We did not include the time taken for common part extraction because, if necessary, this can be executed when time is available such as at night.

The time for the five human subjects were also taken. Since they could judge *fail*ed comparisons more quickly than *pass*ed comparisons, we took the percentage of *pass*es and *fail*s into consideration and calculated a weighted average.

Table VI shows the results. This is the average time to compare one regression test execution. Our tool took less than one-tenth the time of the human subjects.

3) RQ3: Repeat Count: We executed our tool while changing the value of n from 1 to 6. Table VII shows the results. The value indicates the percentage of variable regions that our tool correctly identified. As expected, when the pre-modification test is executed more (i.e., the value of n is greater), the percentage of correct identfication becomes higher.

D. Discussion

1) RQ1: Accuracy: Our approach had an accuracy of over 99%, which is comparable to human beings. The issues our approach had were as follows:

- Warning dialogs
  Five out of the eight incorrect judgements were due to not being able to capture the image of a warning dialog. For example, in Zen Cart, when there is an issue with registering user information, a warning dialog pops up. Our tool uses Selenium to capture images, but these pop up dialogs cannot be captured by Selenium. Thus our current implementation takes an ad hoc approach of using the "Print Screen" function of Windows. Unfortunately, when we were conducting pre-modification test, this method of capturing the dialog failed once. This led to the warning image to be handled as a variable region. In other words, it was not considered as a region image to be checked, and thus led that to be PASS instead of FAIL.
- Change in image size
  Two out of the eight incorrect judgements were due to the image size becoming smaller by two pixels. Whenever there is a difference in image size, our tool first trims the larger image so that the images are of the same size. But our tool did not correctly trim an image.
- Timing of capturing a screenshot
  The final incorrect judgement was due to the timing of capturing a screenshot by Selenium. Our tool currently waits three seconds after the Web page is loaded. The image of the Web page is then captured. This would handle cases where after the Web page is loaded, a widget is briefly shown, and then disappears. Unfortunately, in one case, a widget did not disappear after three seconds, and thus was captured.

Note that none of the above issues were due to our usage of χ^2 histogram distance for the region comparison. Since this approach only considers the distribution of the colors that are used in the Web page, technically there is a possibility that two completely different looking Web pages will have a χ^2 histogram distance of zero, i.e., those two pages are computed to be the same. This did not occur in our evaluation. Of course, more experiments are necessary to correctly conclude that χ^2 histogram distance is sufficient for our approach.

Furthermore, note that for Takai, the accuracy for manual judgement was much lower than the other two applications. This was due to a very small change in the Web page that the subjects were not able to detect. Specifically, in one case, the color of some text in a very small region changed from gray to black. Our tool was able to detect such cases, as the χ^2 histogram distance focuses on color.

2) RQ2: Execution time: As expected, our tool was faster than manual checking. For Zen Cart and Welcart, our tool was 11.5 times faster, while it was 17.4 times faster for Takai. The reason that our tool especially performed better for Takai was probably because Takai's Web page was more complex than the other two, and thus it took subjects more time.

3) RQ3: Repeat count: The result of RQ3 found that the percentage of variable regions that are correctly identified is higher when the value of n (the number of times the pre-modification test is executed) is higher. We consider this from a probabilistic perspective.

Suppose that there are m variable regions in a Web applica-

TABLE VII
PERCENTAGE OF CORRECT IDENTIFICATION OF VARIABLE REGIONS

	n					
	1	2	3	4	5	6
Zen Cart	0	50.0	90.0	90.0	100	100
Takai	0	96.7	100	100	100	100
Welcart	0	76.7	96.7	100	100	100

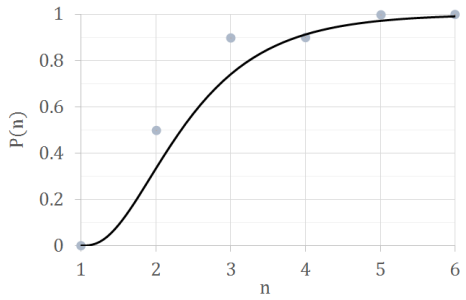

Fig. 4. Repeat time and Accuracy Rate

tion under test and that the pre-modification test was conducted n times. The probability $P(n)$ where the tool can correctly identify all m variable regions can be calculated as shown in equation (2).

$$P(n) = \prod_{k=1}^{m} \left\{ 1 \quad -(\frac{1}{a_k})^{n-1} \right\} \tag{2}$$

In equation (2), a_k means that the kth variable region has a_k contents that can be shown. In other words, for a given instance of a Web page, one of a_k possible contents is shown for the kth variable region. When pre-modification test is executed n times, the probability that the kth variable region is not removed (i.e., correctly identified) is the same as the same content being displayed consecutively n times. Assuming that the probability of one of the a_k contents being shown is the same, a content being displayed consecutively n times can be computed as $\left(\frac{1}{a_k}\right)^{n-1}$. Thus the probability of the kth variable region being correctly identified is a complementary event and can be computed as $1-\left(\frac{1}{a_k}\right)^{n-1}$. Since the number of variable regions is m, we take the product of each variable region resulting in equation (2).

We consider the result of Zen Cart (Table VII), which had 6 variable regions. Of the 6 regions, 3 regions had an extremely high number of contents (i.e., the number of a_k was very large), and thus they can be ignored in terms of equation (2). As for the other 3 regions, the number of contents were 3, 3, and 4. Thus, if we assign $m = 3, \quad a_1 = 3, \quad a_2 = 3, \quad a_3 = 4$ to equation (2), we obtain the curve in Fig. 4. This figure also contains the plots from Table VII.

E. Threats to Validity

The first threat to validity would be the use of students as the human subjects. Since the task was to compare Web page images, which anyone can do, we do not believe that this is a threat in terms of students vs professional. However, there is always the possibility that some humans are better at comparing than others, and we cannot discount this possibility as a threat.

The second threat to validity is the three Web applications we used. We chose the three because they were open source, have variable regions, and have sample data. Evaluation with other Web applications should be conducted to strengthen our findings.

The third threat to validity are the test cases that were used. Test cases were generated using a tool that we had previously developed. We cannot completely deny this as a threat, since other test cases may result in Web pages that are difficult to compare. However, we have manually inspected the possible Web pages, and at least for the three Web applications, we do not believe that the generated test cases would be a threat.

The fourth threat to validity is the values we specified for the parameters REPEAT, SIZE, DISTANCE, and INTERVAL. Table VII showed that specifying the value of REPEAT as 6 had no effect on the result as the correct identification was 100% in all three Web applications. If anything, the value of REPEAT could have been set lower. The value for INTERVAL also had no effect as the three Web applications did not have any variable regions based on the amount of time needing to pass. The values for the other two parameters SIZE and DISTANCE were determined based on preliminary investigation, and these two may pose as threats. In fact, DISTANCE especially seems to be important. This is because if we used the default DISTANCE value of 10 (rather than the 15 we used in the evaluation), differences of 1 pixel would occur when dividing screenshots into regions. Although this seems to be a very small difference, this would lead to incorrectly identifying same images as different. In other words, regions that should be considered as the same would be considered as variable regions. Further evaluation should be done to investigate this.

V. Conclusion

We proposed an oracle for regression testing of Web applications. The oracle consisted of two parts: the expected result and the comparator. The key part of our oracle is accounting for variable regions, i.e., regions within the output Web page that may change each time it appears. The expected result is generated by executing the test multiple times so that variable regions can be identified and removed. The comparator checks the post-modification test execution result against the expected result.

An evaluation of our approach showed that our tool can identify the variable regions as accurately as, but quicker than, human subjects. We also showed that by repetitive execution of the pre-modification Web application improves the accuracy of the comparison.

As for future work, first we need to consider the optimization of thresholds. Our tool uses two thresholds: the minimum size of elements in the DOM tree to specify the region in each screenshot, and the χ^2 histogram distance threshold to determine if two regions should be considered as the same or not. The current (default) values are based on experience from applying our approach to several Web pages. But this may not always be optimal for other Web applications.

Second, although our approach had a very high accuracy, it was still not 100%. One very important issue that we need

to handle to raise the accuracy is to be able to handle dialogs that pop up as the result of executing a test case.

Finally, shortening the execution time further is another important part of future work.

REFERENCES

[1] A. Marback, H. Do, and N. Ehresmann, "An effective regression testing approach for PHP Web applications," in *Proc. IEEE 5th International Conference on Software Testing, Verification, and Validation (ICST 2012)*, 2012, pp. 221–230.

[2] S. N. A. Kamalzaman, S. M. Syed-Mohamad, S. Sulaiman, and K. Zamli, "Supporting maintenance of web applications using user-centered technique," in *Proc. 19th Asia-Pacific Software Engineering Conference*, 2012, pp. 43–49.

[3] L. Xu, B. Xu, Z. Chen, J. Jiang, and H. Chen, "Regression testing for web applications based on slicing," in *Proc. 27th Annual International Computer Software and Applications Conference (COMPSAC 2003)*, 2003, pp. 652–656.

[4] K. Dobolyi and W. Weimer, "Harnessing web-based application similarities to aid in regression testing," in *Proc. IEEE 20th International Symposium on Software Reliability Engineering (ISSRE2009)*, 2009, pp. 71–80.

[5] M. Hirzel, "Selective regression testing for web applications created with Google Web Toolkit," in *Proc. 2014 International Conference on Principles and Practices of Programming on the Java platform: Virtual machines, Languages, and Tools (PPPJ '14)*, 2014, pp. 110–121.

[6] D. Garg, A. Datta, and T. French, "A two-level prioritization approach for regression testing of web applications," in *Proc. 19th Asia-Pacific Software Engineering Conference (APSEC 2012)*, 2012, pp. 150–153.

[7] S. Mirshokraie and A. Mesbah, "JSART: JavaScript assertion-based regression testing," in *Proc. 12th International Conference on Web Engineering (ICWE 2012)*, 2012, pp. 238–252.

[8] S. Raina and A. P. Agarwal, "An automated tool for regression testing in web applications," *ACM SIGSOFT Software Engineering Notes*, vol. 38, no. 4, pp. 1–4, 2013.

[9] V. Dallmeier, M. Burger, T. Orth, and A. Zeller, "WebMate: A tool for testing web 2.0 applications," in *Proc. Workshop on JavaScript Tools (JSTools 12)*, 2012, pp. 11–15.

[10] E. Selay, Z. Q. Zhou, and J. Zou, "Adaptive random testing for image comparison in regression web testing," in *Proc. 2014 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, 2014, pp. 1–7.

[11] A. Stocco, M. Leotta, F. Ricca, and P. Tonella, "PESTO: A tool for migrating DOM-based to visual web tests," in *Proc. 14th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2014)*, 2014, pp. 65–70.

[12] A. Memon, I. Banerjee, and A. Nagarajan, "What test oracle should I use for effective GUI testing," in *Proc. 18th IEEE International Conference on Automated Software Engineering (ASE 2003)*, 2003, pp. 1–10.

[13] S. Choudhary, H. Versee, and A. Orso, "A cross-browser web application testing tool," in *Proc. 26th IEEE International Conference on Software Maintenance (ICSM 2010)*, 2010, pp. 1–6.

[14] "PhantomCSS," 2013. [Online].
Available: https://github.com/Huddle/PhantomCSS

[15] "Applitools." [Online]. Available: https://applitools.com/

[16] "Screenster." [Online].
Available: http://www.creamtec.com/products/screenster/

[17] "W3C Document Object Model." [Online].
Available: http://www.w3.org/DOM

[18] R. Lacanienta, S. Takada, H. Tanno, and M. Oinuma, "A knowledge-based approach for generating test scenarios for web applications," in *Proc. 25th International Conference on Software Engineering and Knowledge Engineering (SEKE 2013)*, 2013, pp. 166–171.

[19] H. Saito, S. Takada, H. Tanno, and M. Oinuma, "Test data generation for web applications: A constraint and knowledge-based approach," in *Proc. 26th International Conference on Software Engineering and Knowledge Engineering (SEKE 2014)*, 2014, pp. 110–114.

[20] "Selenium." [Online]. Available: http://www.seleniumhq.org/

[21] Zen Ventures, LLC, "Zen Cart." [Online].
Available: http://www.zen-cart.com/

[22] JoomlaWorks Ltd., "Takai." [Online]. Available:
http://www.joomlaworks.net/joomla-templates/free-templates/takai

[23] Collne Inc., "Welcart." [Online]. Available: http://www.welcart.com/

[24] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Trans. on Software Engineering*, vol. 37, no. 5, pp. 649–678, 2011.