# Adaptive Random Testing for Image Comparison in Regression Web Testing

Elmin Selay *
School of Computer Science and Software Engineering
University of Wollongong
NSW 2522, Australia

Zhi Quan Zhou **
School of Computer Science and Software Engineering
University of Wollongong
NSW 2522, Australia

Jingjie Zou
College of Computer & Communication Engineering
China University of Petroleum
Shandong, P.R. China

***Abstract* — Web applications have become the most popular type of software in the past decade, attracting the attention of both the academia and the industry. In parallel with their popularity, the complexity of aesthetics and functionality of web applications have also increased significantly, creating a big challenge for maintenance and cross-browser compliance testing. Since such testing and verification activities require visual analysis, web application testing has not been sufficiently automated. In this paper, we propose a novel pairwise image comparison approach suitable for web application testing where the location of layout faults needs to be detected efficiently while insignificant variations being neglected. This technique is developed based on the characteristics of fault patterns of browser layouts. An empirical study conducted with the industry partner shows our approach is more effective and efficient than existing methods in this area.**

***Keywords — adaptive random testing; image comparison; layout faults; random testing; regression testing; Web applications.***

## I. INTRODUCTION

Nearly 20 years after the first ping was sent via what we call today as the Internet, Tim Berners-Lee invented the World Wide Web (WWW), which not only reshaped how information is communicated but also changed the mankind irreversibly in their real and virtual worlds. With explosive adoption in the past two and half decades, the number of regular internet users reaches 3 billion in the world as of 2014 [1]. As a result, web applications have become the most used software products, attracting more and more attention from both the industry and the academia. Although the sole purpose of a web page was to deliver information in a simple format two decades ago, the complexity of aesthetics and functionality of web applications has increased almost in parallel, making development, maintenance and testing tasks much harder [2,3]. The main purpose of such testing activities is to make sure that the target application not only functions consistently but also appears consistent for the target audience.

In order to identify layout faults, human testers must inspect each web page by eye to verify correctness even in most automated scenarios because the standard-compliant code of the web application does not guarantee its correct look and feel. This manual work is time-consuming and expensive and multiplies given that testing is done repeatedly over time (regression testing) when a new enhancement, patch or fix is made to the already tested application [4, 5].

The quantity and complexity of test cases increase at each stage since the tester would ideally run all previous test cases besides new test cases when such an enhancement is made to make sure that the new feature does not affect existing functionality.

One of the main challenges in web testing automation attempts has been an effective visual analysis of layout images captured from different browsers by advanced testing tools such as Selenium [6]. Since screenshots are large in size and expected to be different to certain extent even when no bugs exist, the requirement for an effective image comparison algorithm is to discover the location of a layout fault, if any, efficiently at a low computational cost by neglecting insignificant details to avoid false positive cases. Just as this approach is different from traditional image matching, pattern recognition, histogram similarity/difference analysis, so too it is poles apart from perceptual comparison methods.

Our proposed image comparison method is a step to overcome this shortcoming. The results of the experiments are encouraging and proved to be effective in an empirical study to be presented in this paper. The remaining of the paper is organized as follows: Section II explains the background to the problem and defines the research question. Section III reviews related work in the field of web application testing. Section IV presents our approach and its internal working mechanism. Section V presents an empirical study on the proposed image comparison method and its effective layout fault detection. Section VI concludes the paper.

* Elmin Selay is also with Tickets.com Pty Ltd, Australia.
** All correspondence should be addressed to Dr. Zhi Quan Zhou, School of Computer Science and Software Engineering, University of Wollongong, Wollongong, NSW 2522, Australia. Email: zhiquan@uow.edu.au.

978-1-4799-5409-4/14/$31.00 ©2014 European Union

## II. BACKGROUND

### A. Web browsers

A web application is run inside a web browser on client's machine which can be from mobile phones, tablets, desktop computers to any internet-enabled devices. In addition to its all extra features, the primary function of a browser is to render the visual output by interpreting the simple markup directives and auxiliary style and control instructions [7].

After web technologies started to be commercialized in early 1990s, several variations of web browsers appeared with their own extended features and layout rendering engines. The World Wide Web Consortium (W3C) was established in 1994 to regulate interoperability and compliance among browsers. Despite all attempts, the full standardization and interoperability has never been achieved within the past two decades. Assuming application developer cannot restrict which browser or device users will use to access the application, the developer must do significant amount of cross-browser and cross-device testing to achieve consistency.

### B. Software testing

Among many types of software testing, regression testing plays an important role in software maintenance where the purpose is to ensure that a new enhancement to the existing application does not break any previously tested parts [4,5].

Testing requires two key test components: test data (or test cases) and test oracle - where the former mainly refer to the inputs given to the application while the latter is the mechanism against which the result is checked to determine if it is correct [8]. Although human oracles often verify the correctness of the actual result, automated software testing oracles have been classified as complex and expensive to develop since it involves extracting an oracle from requirements specification, program simulation or a trusted implementation [2,8].

Unlike other software, web applications produce its output in visual graphical form, complicating the verification for non-human test components. This paper mostly focuses on the very visual analysis automation to facilitate such verification.

### C. Motivating example and problem statement

We made a simple example to illustrate the relevance of the problem and motivation behind the automation of the visual analysis in web application testing. The following sample html and CSS code snippet validated by the W3C Validation Service as *standard-complaint* renders completely different output in different browsers:

```
<!DOCTYPE html>
<html lang="en-AU">
<head>
   <style type="text/css">
    body{ padding:10px;}
   .box {width:250px;height:250px;float:left;border-radius:50%;overflow:hidden;
         transform:skew(10deg,15deg);border:solid 4px black; box-shadow: 0 0 0 8px #ff0030;
         color:black; text-align:center; font-size:2.2rem;line-height:250px; }
   .box.blue-red {background: linear-gradient(red, yellow, blue);content: "Hello";
         transform: rotate(120deg);letter-spacing: 4px;}
   .box.blue-red span:after{content:"World";}
   .box:first-child {margin-right:40px;}
   img { height: 100%;width: 100%; object-fit: contain;}
   </style>
</head>
<body>
   <div id="main">
      <div class="box">
         <img src="http://www.bubblews.com/assets/images/news/619314055_1366210644.jpg" />
      </div>
      <div class="box blue-red">
       <span>Hello</span>
       </div>
   </div>
</body>
</html>
```


**Figure 1: Google Chrome Version 35.0.1916.114 m**

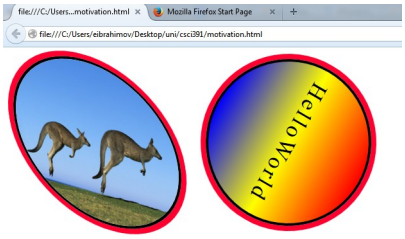


**Figure 2: Mozilla Firefox Version 29.0.1**

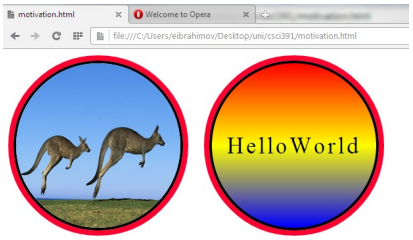


**Figure 3: Opera Version 17.0.1241.45**


**Figure 4: Safari 5.1.7 (7534.57.2)**

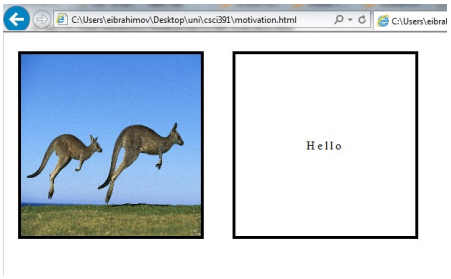


**Figure 5: Microsoft Internet Explorer 7 (Mode)**


**Figure 6: 360 (Chinese) browser Version 6.3**

As shown, a simple code snippet fully compliant with the standards renders a completely different output in different browsers (Figures 1 to 6). This example also illustrates that unlike other software products, standard-compliant and verified code does not guarantee the expected look and feel of the web applications. Therefore, testing tools which verify correctness of markup syntax and structure cannot verify the correctness of the output in multiple platforms and browsers (and their different versions).

To make the same markup code to render exactly the same output in all browsers requires a huge amount of development and testing time and costs. Given the amount of code in a medium-sized fully functional web application, it is not hard to imagine the amount of efforts needed. As web applications change more frequently than other types of software, each enhancement or feature added to the existing software requires regression testing to make sure the new extension does not cause issues with any previously tested and functioning units.

This motivation example substantiates the necessity of effective automation of visual analysis of web application interfaces to minimize human-tester oracle involvement and judgment. As regression testing involves previous test cases, there are sufficient self-oracles – results previously rendered on the same browser/platforms or results rendered on different browser/platforms to compare for verification.

### *D. Research question*

Although there have been extensive work on development of scenarios and tools for web application testing by both the industry and research community, all of these approaches either consequently rely on human oracles or lead to unnecessarily high computational cost in comparison, as outlined in Section III. Since there already exist a large number of easily-configurable automated testing tools such as Selenium to capture web application output as screenshots, this research mainly focuses on proposing an efficient output comparison method. This research question is stated as follows:

*Is it possible to programmatically compare web application output screenshots at a low computational cost to identify layout faults by neglecting insignificant variations?*

Before giving a detailed answer to the above research question, some related work is reviewed in the next section.

## III. RELATED WORK

There has been significant work by the academia and the industry towards automating software testing. As the focus of this research is web applications, this section only reviews closely related work done in the relevant area.

The current state of the art used in the industry is a manual and visual evaluation of a web page to determine whether the rendered output is correct or acceptable. Two popular technological solutions to automate web testing in the past decade have been Microsoft Expression Web [9] and Adobe's Browser Lab [10]. Although the former validates the standards-compliance of the markup code without providing any information about how it looks or behaves in different browsers or screen sizes, the latter, which was discontinued by Adobe from May 2013, provides screenshots and diagnostic tools, leaving it to the developer to make the decision. Nowadays Selenium is the most commonly used tool to programmatically automate screenshot capturing process. By not addressing the issue of automated testing, these tools require the developer to spend a significant amount of time in visually evaluating and understanding what the issue is. There have been a score of similar commercial tools or online testing applications available with no easy-to-use configuration options.

Literature until mid-2000s mostly focused on the white-box testing of web applications mostly due to the facts that web testing was a new notion and Microsoft Internet Explorer had the lion's share in the browser market and there was little concern of multi-browser compatibility issues. Eaton and Memon [11] proposed a compliance assessment technique, which requires the developers to manually provide examples of correct and incorrect pages for the calculation of probability of next tag's being faulty. A limitation of this solution is that it does not consider Cascading Style Sheets (CSS), which provides layout and style to an html document. As Web technologies started to gain more popularity thanks to the social media boom and a surge in the internet penetration rate across the globe, automation of web testing attracted more attention from the academia and industry.

Choudhary et al. proposed a new method using WebDiff to automate the identification of cross-browser issues [2]. Their method is proposed to open a web page in browsers, read the Document Object Model (DOM) information and take a screenshot and compare attributes of the document nodes to find inconsistencies. They also implemented a histogram based comparison of screenshots. This approach has also made attempts to improve histogram comparison side effects as further discussed in Section IV. With its innovations considered, this approach was the most significant step towards automation of web application testing.

Mesbah and Prasad proposed a new method of automation in this area [12]. Their approach used open-source Ajax crawler Crawljax to capture and store the observed behaviour of the web page for each browser as what they called finite-state machine navigation model. In the second step, they compared generated models to find discrepancies. Such comparison or equivalence checking is done by extracting state graphs from the navigation models and comparing edges

and nodes. For this purpose, each node represents an abstraction of the DOM tree instance. Although the solution automates testing process, the data collected and comparison is done on the DOM level.

## IV. EFFECTIVE IMAGE COMPARISION FOR LAYOUT FAULT DETECTION

For the past decades, image matching has remained as an important challenge for computer vision and automation of visual analysis for equivalence check. Existing image matching algorithms either compare the images for difference, similarity or identity or use sophisticated approaches of perception metrics or pattern recognition. Although there are many open source tools implementing image comparison algorithms, none of them suits the needs of web application testing.

Comparing images by their histograms where the histogram data present the distribution of the feature in the image leads to a bin-to-bin or cross-bin comparison. Such an approach has been used in comparing web application images by Chaudhary et al. [2]. To overcome the false positive cases in a basic histogram matching, Chaudhary et al. used the Earth Movers' Distance (EMD) to mitigate the problem. The main purpose of using EMD in computer vision is to compare discrete distributions, which neglect small variations in shape matching, image retrieval, texture analysis and object recognition by measuring the distance between two probability distributions across a region – corresponding to Wasserstein metric [13,14,15].

For most histogram similarity measurements, the time needed in calculating the similarity between two different histograms is proportional to the size of the histograms. This scheme has been successfully used in different areas of the industry. For example, a novel class of quadratic form distance functions for assessing the similarity of colour histograms is currently used in IBM's QBIC image retrieval system [14]. Although this type of approaches has wide-spread usage in the industry, they have their own application area. Additionally, a main concern of this solution is its time complexity.

Without doubting the effectiveness and power of EMD algorithms, it must be noted for browser layout image matching with special characteristics, there is a need for a cheap, easy-to-implement and effective solution. For example, there is no need to identify objects or patterns with precision. In order to create a background picture for this necessity, it is important to look at the failure patterns.

### A. *Failure patterns*

It has been observed that there are typical distribution patterns of failure-causing inputs, namely, block pattern, strip pattern, and point pattern as in Figure 7 [21, 26], where the block and strip patterns can be very common in real-world applications.

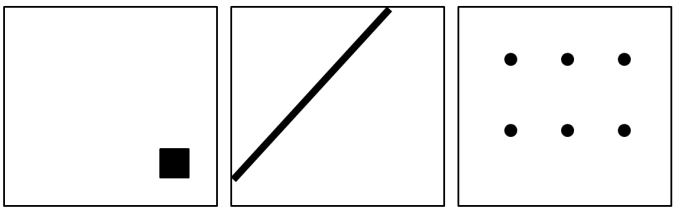

**Figure 7: Failure patterns: block pattern (left), strip pattern (middle) and point pattern (right)**

Considering these observations, we can observe the failure patterns of browser layout. As defined in W3C standards [16], elements of the web document are rectangular boxes laid out one after another or nested inside each other. This type of ordering is called a flow. The flow means "by default, elements flow one after another in the same order as they appear in the HTML source, with each element having a size and position that depends on the type of element, the contents of the element, and the display context for the element as it will render on the page" [17]. Since the elements are rendered as rectangular shape but masked or their border radius is changed from inside to have different shapes, we can intuitively assume that the potential failure pattern the faulty elements can create is likely to be block failure patterns. Also, as elements flow one after another, a faulty element rendered wrongly will push the next element to flow, still creating block failure patterns (such as domino effect) as it may drop to the next row if not fitting (this further reduces the chance of strip patterns). Point patterns should be rarer and, even if existing, they each will cluster to form block pattern as to the statement above.

Leveraging this application domain knowledge, we can select a suitable approach for equivalence checking. The approach selected must eliminate the need for checking all possible colour histograms, possibly making sure the checked regions are apart from each other. This is because block failures in browser layouts are more likely to cause a domino effect. As a result, the target is to hit the first failure region in an effective way. However, to do this, we need to explore the distribution of the test case generation.

### B. *Adaptive random testing*

One of the simplest ways to distribute test cases is to use random testing approach to check only the random points in the captured screenshot. Random testing is simple and cheap to implement and effective to detect failures and allows generating input in unexpected ways and ranges to discover the fault [18,19,20].

Although random testing has a low cost and is widely used, it does not use any available information to adapt to the needs. To improve this while keeping the good feature of randomness, an adaptive random testing (ART) strategy has been developed [21]. Unlike pure random testing, ART makes use of the knowledge of previously executed test cases so that the next test cases can be selected in such a way that they are more evenly distributed over the input domain. This concept of even distribution leverages the fact of common continuous failure patterns to reduce the number of test cases needed to detect a failure [21,22].

A widely adopted ART algorithm is known as the Fixed Size Candidate Set (FSCS) ART algorithm [21], which works as follows: the first test case is randomly selected. Then, every time a new test case is needed, a set of test case candidates are generated. This set has a fixed size (e.g. 10). For each candidate, its distances to all the previously executed test cases are measured and the shortest distance is recorded. The candidate having the longest shortest distance is selected to be the next test case and all the other candidates are discarded. This process is repeated until a stopping criterion is met (such as when a failure is detected). Compared with random testing, test cases generated by FSCS ART are farther apart from each other and they still reserve the desirable property of randomness. The FSCS ART algorithm has an $O(n^2)$ time complexity, where n is the number of test cases generated. Much research has been done to improve the time complexity of ART, and one such solution is known as "forgetting test cases" [23]. Instead of remembering all of the previously executed test cases, a "forgetting" algorithm can choose to remember only a small and fixed number of previously executed test cases. In this way, an FSCS-ART-with-Forgetting algorithm has a linear time complexity of O(n), which is in the same order of complexity as random testing.

ART's test case generation time can also be reduced by means of the mirroring technique [24], which reflects or translates the generated test cases in one sub-region of the input domain to other sub-regions at a very low cost.

### C. Our image comparison method

As images represent grid-based objects, we propose application of a FSCS ART with forgetting algorithm to create a fair distribution of pairwise image checkpoints. The screenshot layout is divided into multiple regions and a single ART point generated in one of the regions is translated to all regions without further calculation. Calculating translation is cheap and easy-to-implement because the point on the coordinate system only changes x value to be translated in the region aligned horizontally with it, and only changes y value to be reflected in the region aligned vertically with it. Such an example is shown in Figure 8.

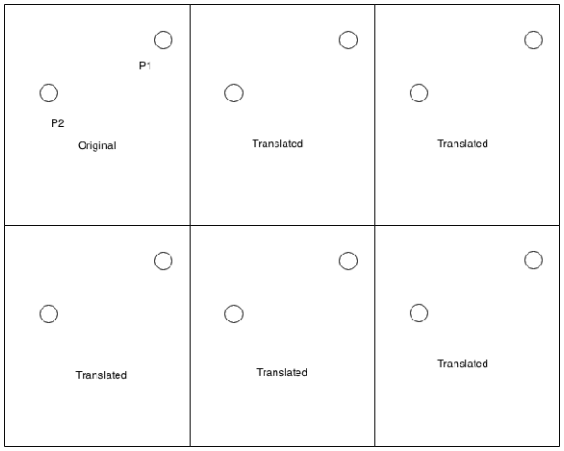


**Figure 8: Mirroring P1 and P2 in neighbouring regions**

To the best of the authors' knowledge, the present research is the first work that applies ART to image comparison. Using this method leverages the application domain information and the fact of existence of contiguous failure regions and block failure patterns in browser layout fault occurrences. The main advantage of this method is that it keeps the power of random testing while the coverage area grows faster thanks to the even spread of test cases generated by ART.

To investigate the effectiveness of our image comparison method, an empirical study (to be outlined in the next section) has been conducted using four approaches: sequential (brute force – comparing all pixels of the two dimensional image from left to right and top down), random, FSCS-ART-with-Forgetting (referred to as ART-forgetting), and FSCS-ART-with-Forgetting-and-Mirroring (referred to as ART-forgetting-mirroring).

## V. EMPIRICAL STUDY

To evaluate the effectiveness of the proposed solution, a set of real world web applications from the industry partner (Tickets.com Pty Ltd.) were tested. Staging and production versions of web applications have been used for pairwise comparisons.

### A. Hardware and software

The testing was conducted on Windows 7 Enterprise Edition with 4 most popular web browsers (Google Chrome Version 35.0.1916.114m, Mozilla Firefox Version 29.0.1, Opera Version 17.0.1241.45 and Microsoft Internet Explorer 10.0.15) involving 1,300 pairs of web pages from 7 real world web applications. For each pair of web page, say, (A, B), where A is the previously tested version of the site (production version) and B is the staging version (a new version of the site) under test, we have the screenshot of A only on the reference browser (e.g. Firefox), and run B on 4 browsers. B's screenshots from the 4 browsers are taken and are compared to the screenshot of A.

### B. Testing framework

In order to programmatically feed the image comparison module with screenshots of web application output from a certain browser, a WebDriver is required. A browser WebDirver provides an API functions which can be called programmatically to simulate user behaviour such as loading pages and clicking or invoking other functionalities of the web application. There are several implementations of such WebDrivers by both the open-source community and the official browser vendors. We used Selenium WebDriver for the empirical study because of the popularity and acceptance rate among the web application development community.

### C. Testing environment setup

The testing process involved a simple user-defined configuration file (to set browsers, exclusion zones, such as ad banner classes to neutralize them, application-specific paths etc.), a driver program (controller) to read the configuration file to feed the browser WebDriver (as part of Selenium) and record the output and call the comparison module under

evaluation to carry out pairwise comparison. The controller also records a set of metrics such as execution time and F-measure (number of test cases used to detect the first failure).

It is to be noted that, by allowing users to set the exclusion zones, our testing tool can ignore the differences in these exclusion zones – these differences are regarded as "insignificant variations" stated in our research question raised in Section II-D.

### D. Metrics and measurement

The fault detecting strategy in testing is characterized by its efficiency and effectiveness (in terms of the cost, time and resources). This is because the testing cannot be exhaustive and the conduct of testing to a reasonable extent is expected to either discover the faults or end without detecting any.

In order to evaluate the failure-detection capability of a testing method, researchers have developed several metrics. The three most common metrics for effectiveness evaluation are: P-measure: probability of finding at least one failure by the set of test cases run; E-measure: expected number of failures detected by the set of test cases run; and F-measure: expected number of test cases required to discover the first failure [18,22,25,26]. The sampling spread of P-measure and E-measure draw very close in normal distribution, while the probability distribution of the F-measure turns out to be geometric [18].

Without going into details, we need to answer which of these measures best describe the testing effectiveness. The answer is it depends on the constraints, conditions and nature of the testing done. Considering the browser layout faults fall into the category of the block failure pattern with defective zones clumped together, detecting multiple failures can be redundant because the first failure is the most important discovery. Also, in real world application, the tester may not be interested to get dozens of failure regions detected in the same page because the first failure region will attract the attention and will return the control to the developer. This approach also illustrates the importance of F-measure and P-measure over E-measure. Therefore, for the purpose of this testing process, F-measure, P-measure and additionally execution time (ms) to the first failure are recorded. Execution time is usually an interesting parameter as it tells how quickly the failure can be detected. However, execution time is platform and environment dependent.

### E. Results of experiments

The test results and measurements have been done in accordance with the metrics and measurement outlined above. For each of the 1,300 pairs of web pages from the 7 real world web applications, each page pair (production page, staging page) was tested for 100 times using random testing, 100 times using ART-forgetting, and 100 times using ART-forgetting-mirroring. Note that each time the production site was run on one browser but the staging site was run on all 4 browsers to collect screenshots. Therefore, the scale of the experiments is nontrivial. All F-measures were collected and the average F-measure is calculated for each algorithm. The sequential brute force algorithm need only be run once instead of 100 times as it always gives the same sequence. To calculate the P-measure, 1,000 test cases/pixels are applied each time. For the two ART with forgetting algorithms, only the 10 most recently executed test cases are remembered. For the mirroring strategy, we divided each image into 4 sub-regions.

**Table 1: Results of experiments**

| Method | Mean time to first failure (ms) | Mean F-measure | P-measure |
|---|---|---|---|
| sequential brute force | 6,381 | 175,794 | N/A |
| random | 110 | 18.01 | 0.24 |
| ART-forgetting | 112 | 18.41 | 0.24 |
| ART-forgetting-mirroring | 106 | 15.10 | 0.24 |

Table 1 summarizes the experimental results. The sequential brute force algorithm is used as benchmark and experimental control. This algorithm is obviously the worst among all 4 algorithms. This finding indicates that comparing images pixel by pixel sequentially is highly cost-ineffective in terms of finding a failure quickly.

ART-forgetting-mirroring is found to be the best algorithm in this study: It has the shortest mean time to first failure and the smallest mean F-measure. Random testing outperformed ART-forgetting in execution time and F-measure, but the difference is not large. All 3 algorithms had similar P-measures, after rounding to two decimal points. In addition, the discovery of false positives obviously decline drastically in proportion since the number of test cases/pixels are reduced in both random and ART methods.

## VI. Conclusion

Automating the visual analysis of testing web applications has been a challenge for the past two and half decades. Despite some initiatives from the research community and the industry, the approaches have been immature and have never been widely accepted by the industry. While one of the reasons is the complexity of the nature of web application testing, the other reasons include the efficiency and cost effectiveness. In this research, practical experiments confirmed our approach as a promising method towards automating web application testing as the proposed layout screenshot comparison method is shown to outperform existing solutions in its effective and efficient fault detection capability. The decline in the discovery of false positive cases is also a significant factor because it reduces the probability of false alarms due to the insignificant variations, which can be adjusted by the web developers by setting certain fault tolerance limits and/or regions.

The proposed approach gives a good answer to the research question raised in Section II of this paper. It minimizes the amount of manual work required from testers and developers during regression testing of web applications. This method is independent from web application testing and

can also be applied to other areas such as automation of visual analysis in medical imaging and robotic surveillance/observation vision systems.

There are several possible areas for future work. We plan to apply several more domain-specific knowledge leverages to further improve the image comparison method's effectiveness. We also plan to increase the scale of the empirical studies.

## References


[1] UN International Telecommunications Union, "ITU releases 2014 ICT figures," http://www.itu.int/net/pressoffice/press_releases/2014/23.aspx, accessed in May 2014

[2] S. Roy Choudhary, H. Versee, and A. Orso, "Webdiff: Automated identification of cross-browser issues in web applications," in *Proceedings of the 26th IEEE International Conference on Software Maintenance* (*ICSM'10*), pp.1—10, 2010.

[3] D. Robins and J. Holmes, "Aesthetics and credibility in web site design," *Journal of Information Processing & Management*, vol. 44, no 1, pp. 386—399, 2008.

[4] G. Rothermel, R. H. Untch, C. Chengyun, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 929—948, 2001.

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

[6] OpenQA, "Selenium web application testing system," http://seleniumhq .org, accessed in May 2014.

[7] A. Grosskurth and M. W. Godfrey, "A reference architecture for web browsers," in *Proceedings of the 21st IEEE International Conference on Software Maintenance* (*ICSM'05*), pp. 661–664, 2005.

[8] D. J. Richardson, S. L. Aha, and T. O. O'Malley, "Specification-based test oracles for reactive systems," in *Proceedings of the 14th International Conference on Software Engineering* (*ICSE'92*), pp. 105–118, 1992.

[9] Microsoft Corporation, "Expression web," http://www.microsoft.com /expression/, accessed in May 2014.

[10] Adobe Systems Incorporated, "Browser lab," http://blogs.adobe.com /browserlab/, accessed in July 2014.

[11] C. Eaton and A. M. Memon, "An empirical approach to evaluating web application compliance across diverse client platform configurations," *International Journal of Web Engineering and Technology*, vol. 3, no. 3, pp. 227–253, 2007.

[12] A. Mesbah and M.R. Prasad, "Automated cross-browser compatibility testing," in *Proceedings of the 33rd International Conference on Software Engineering* (*ICSE'11*), pp. 561—570, 2011.

[13] 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.

[14] M. Flickner, H.Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D.Steele, and P.Yanker, "Query by image and video content: the QBIC system," *Computer*, vol. 28, no. 9, pp.23—32, 1995.

[15] L. Ling and K. Okada, "An efficient earth mover's distance algorithm for robust histogram comparison," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 5, pp. 840—853, 2007.

[16] W3C Specifications, http://www.w3.org/Style/CSS/specs.en.html, accessed in May 2014.

[17] Microsoft Developer Network, http://msdn.microsoft.com/en-us/library/ms533005(v=vs.85).aspx, accessed in July 2014.

[18] T. Y. Chen, F.-C. Kuo, and R. Merkel, "On the statistical properties of the F-measure," in *Proceedings of the 4th International Conference on Quality Software* (*QSIC'04*), pp. 146—153, 2004.

[19] T. Y. Chen and R. Merkel, "Quasi-random testing," *IEEE Transactions on Reliability*, vol. 56, no. 3, pp. 562—568, 2007.

[20] S. Morasca and S. Serra-Capizzano, "On the analytical comparison of testing techniques," in *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis* (*ISSTA'04*), pp. 154–164, 2004.

[21] T. Y. Chen, F.-C. Kuo, R. Merkel, and T. H. Tse, "Adaptive random testing: The ART of test case diversity," *Journal of Systems and Software*, vol. 83, no. 1, pp. 60–66, 2010.

[22] F. T. Chan, T. Y. Chen, I. K. Mak, and Y. T. Yu, "Proportional sampling strategy: guidelines for software testing practitioners," *Information and Software Technology*, vol. 38, no. 12, pp.775—782, 1996.

[23] K. P. Chan, T. Y. Chen, and D. Towey, "Forgetting test cases," in *Proceedings of the 30th Annual International Computer Software and Applications Conference* (*COMPSAC'06*), pp. 485—494, 2006.

[24] T. Y. Chen, F.-C. Kuo, R. Merkel, and S. P. Ng, "Mirror adaptive random testing," *Information and Software Technology*, vol. 46, no. 15, pp. 1001—1010, 2004.

[25] T. Y. Chen and R. Merkel, "An upper bound on software testing effectiveness," *ACM Transactions on Software Enginering and Methodology*, vol. 17, no. 3, pp. 1—27, 2008.

[26] T. Y. Chen, T. H. Tse, and Y. T. Yu., "Proportional sampling strategy: A compendium and some insights," *Journal of Systems and Software*, vol. 58, no. 1, pp. 65–81, 2001.