WebEvo: Taming Web Application Evolution via Detecting Semantic Structure Changes

Fei Shao
Department of Computer and Data Sciences
Case Western Reserve University
United States of America
fxs128@case.edu

Rui Xu
Department of Computer and Data Sciences
Case Western Reserve University
United States of America
rxx100@case.edu

Wasif Haque
Department of Computer Science
University of Texas at Dallas
United States of America
wah180000@utdallas.edu

Jingwei Xu
School of Electronic Engineering and Computer Science
Peking University
China

Ying Zhang
School of Electronic Engineering and Computer Science
Peking University
China
zhangying06@sei.pku.edu.cn

Wei Yang
Department of Computer Science
University of Texas at Dallas
United States of America
wei.yang@utdallas.edu

Yanfang Ye
Department of Computer and Data Sciences
Case Western Reserve University
United States of America
yanfang.ye@case.edu

Xusheng Xiao
Department of Computer and Data Sciences
Case Western Reserve University
United States of America
xusheng.xiao@case.edu

ABSTRACT

The development of Web technology and the beginning of the Big Data era have led to the development of technologies for extracting data from websites, such as information retrieval (IR) and robotic process automation (RPA) tools. As websites are constantly evolving, to prevent these tools from functioning improperly due to website evolution, it is important to monitor the changes in websites and report them to the developers and testers. Existing monitoring tools mainly use DOM-tree based techniques to detect changes in the new web pages. However, these monitoring tools incorrectly report content-based changes (i.e., web content refreshed every time a web page is retrieved) as the changes that will adversely affect the performance of the IR and RPA tools. This results in false warnings since the IR and RPA tools typically consider these changes as expected and retrieve dynamic data from them. Moreover, these monitoring tools cannot identify GUI widget evolution (e.g., moving a button), and thus cannot help the IR and RPA tools adapt to the evolved widgets (e.g., automatic repair of locators for the evolved widgets). To address the limitations of the existing monitoring tools, we propose an approach, WebEvo, that leverages historic pages to identify the DOM elements whose changes are content-based changes, which can be safely ignored when reporting changes in the new web pages. Furthermore, to identify refactoring changes that preserve semantics and appearances of GUI widgets, WebEvo adapts computer vision (CV) techniques to identify the mappings of the GUI widgets from the old web page to the new web page on an element-by-element basis. Empirical evaluations on 13 real-world websites from 9 popular categories demonstrate the superiority of WebEvo over the existing DOM-tree based detection or whole-page visual comparison in terms of both effectiveness and efficiency.

CCS CONCEPTS

• **Software and its engineering** → **Software evolution**; **Software testing and debugging**.

KEYWORDS

Web Application Evolution; DOM-Tree Analysis; Computer Vision

ACM Reference Format:
Fei Shao, Rui Xu, Wasif Haque, Jingwei Xu, Ying Zhang, Wei Yang, Yanfang Ye, and Xusheng Xiao. 2021. WebEvo: Taming Web Application Evolution via Detecting Semantic Structure Changes. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '21), July 11–17, 2021, Virtual, Denmark*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3460319.3464800

Xusheng Xiao is the corresponding author.

ISSTA '21, July 11–17, 2021, Virtual, Denmark

ACM ISBN 978-1-4503-8459-9/21/07…$15.00
https://doi.org/10.1145/3460319.3464800

1 INTRODUCTION

With the development of web technology and the beginning of the Big Data era, websites have become increasingly rich with various types of content, which has led to the development of technologies for extracting data from websites. Several Information Retrieval (IR)

tools are constantly reading data from web pages and using them to power different applications. These tools depend on the location of the data to be extracted on a given web page. The common positioning techniques for elements in web pages include XPath (XML Path Language) positioning [45], CSS (Cascading Style Sheets) selector positioning [46], and other simple positioning techniques based on different attributes. In web applications, the elements on the web pages are often changed due to requirements change or get displaced due to insertion or deletion of other elements. Such changes can cause these positioning techniques to fail to locate the elements to extract data from.

Besides IR tools, robotic process automation (RPA) tools [43] that leverage automated scripts to automate the tasks in the Graphical User Interface (GUI)s of applications are also gaining attentions from industry [1, 44], and web application testing tools [6] also use automated scripts to verify the proper functioning of web applications. The automated scripts used by web testing and RPA automate manual operations performed on the web application's GUI, such as sending click events, filling in and submitting forms. Despite their popularity, such automated tasks and tests are prone to failure due to simple structural changes in a web page. Researchers have identified web element locators to be the main reason for the failure of test scripts [18, 41]. Due to changes in a web page structure, elements on a web page can move to different locations within the page or get deleted, which can cause task failures, where the automated tasks fail to deliver their expected results, and *test breakages* - where the tests raise exceptions that do not reflect the presence of a bug.

As addressing these web application evolution issues by manual maintenance is labor-intensive and time consuming, it is important to develop monitoring tools that can automatically identify the website changes that can potentially stop IR tools and test scripts from functioning properly. Existing monitoring tools [18, 22] mainly adopt DOM-tree based techniques, which detects website changes by analyzing the differences in the DOM structures of the new and the old web pages. Considering only the differences in the DOM of web pages makes the techniques susceptible to high number of false positives, since IR and RPA tools expect some DOM elements to be constantly changed (e.g., changes of weather values) and extract the content from these elements as the new data. Besides DOM-tree based techniques, the state-of-the-art web test repair tool Vista's [41] also leverages computer vision (CV) techniques to identify changed elements that preserve their appearances in new web pages, and repair test scripts to use the updated locators of the changed elements. However, their adopted template-based matching technique [28, 38] compares the screenshots of whole web pages, which is also prone to false positives since the precision of the technique is greatly affected by background colors and the appearance of other web elements that are close to the changed elements.

Towards developing an automated tool for detecting changes in websites, we identify two technical challenges.

- ① The first challenge is determining the type of changes that can cause IR and RPA tools to break and identifying the changes on a web page. We observe that there are two types of web page changes - *content-based changes* and *semantic structure changes*. We define content-based changes as DOM elements whose content being constantly updated based on what a web server delivers to the client browser, such as stock prices, weathers, and advertisements. This type of changes are represented as DOM subtrees rooted in some unchanged elements (e.g., the ** ** that holds the weathers). As IR and RPA tools typically locate these changes using the unchanged elements, content-based changes usually do not cause task failures. As opposed to content-based changes, semantic structure changes refer to element changes that alter the underlying structure of a web page. Such changes to a web page or parts of a web page are the root cause of element locators failing to find new locations of modified or deleted elements. An illustrated example is in Section 2.
- ② The second challenge relates to using CV techniques to perform visual analysis on the web elements. Sometimes an evolved web page may move a link to a different location (e.g., moving to another ** ** element) or change the structure and attributes of a textbox (e.g., updating the **name** attribute) but preserves its semantics and appearance. Such changes often result in the changes of the locators for these elements. It is important to map these changed elements to the original elements in the old web pages, so that IR and RPA tools that operate on the original elements can be automatically fixed by using the mappings. As the structures of the changed elements and the original elements are generally different, DOM-tree based techniques often consider the changed elements as new elements and cannot find the correct mappings. For example, an element that changes its tag from **<p>** to ** ** but keeps other attributes intact (i.e., preserving semantics and appearance) will be identified by DOM-tree based techniques as a new element, producing a false positive and a false negative (i.e., not able to detect the tag update).

To address the above challenges, we propose WebEvo, an automated web monitoring tool that detects DOM elements with semantic structure changes. WebEvo first generates candidates for changed elements by comparing the DOM trees of old and new web pages, including identifying new, updated, and deleted elements. To address the challenge ①, WebEvo leverages historic web pages of the old web page to identify the elements whose text and images are different across historic web pages as content-based changes. The content-based changes are then filtered from the candidate changed elements. To address the challenge ②, rather than analyzing the screenshots of whole web pages, WebEvo obtains the screenshots of the changed elements and combines both text similarities and image similarities to identify the mappings between the elements in the old and new web pages. In this way, our fine-grained analysis that compares the screenshots on an element-by-element basis minimizes the noises brought by background images and nearby elements, and the combination of text similarity and image similarity can correctly create mappings even if many elements in the new web page have similar appearances.

WebEvo is evaluated on 13 real-world websites from 9 popular categories [2–4]. For each website, we choose a target web page, an evolved page, and three history pages to detect changed elements. Our results show that WebEvo is highly effective in detecting the changed elements, achieving an average precision of 0.91, an average recall of 0.79, and an average F_1 score of 0.84. We also

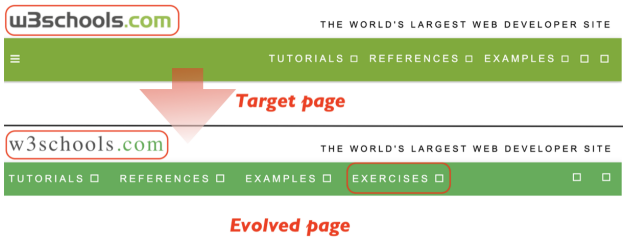


Figure 1: A new link in *www.w3schools.com*

Figure 2: New social links in *www.imdb.com*

compare WebEvo against a DOM-tree based technique [18, 22] and Vista [41], the state-of-the-art visual test repair tool. The results show that WebEvo is significantly more effective than the DOM-tree based technique, which achieves a F_1 score of 0.50. For the comparison with Vista, we compare the detection precision for only the deleted and the updated elements, excluding the added elements. This is because Vista finds the matched elements in the new web pages for the elements in the old web pages, and thus Vista cannot detect added elements in the new web pages. The comparison results show that Vista achieves a F_1 score of 0.39, which is much worse than WebEvo's F_1 score (0.84). Such results demonstrate the superiority of WebEvo over the state-of-the-art techniques. Finally, we also compare the runtime performance of WebEvo and Vista, and the results show that WebEvo's analysis time is averagely 42.01% less than Vista's. This shows that Vista's visual search on the whole web page requires significantly more time than WebEvo's element-wise visual search. The tool and the results are available publicly at the project website [39].

Our paper makes the following contributions:

- WebEvo is the **first technique** that filters content-based changes and detects only semantic structure changes for evolving web pages.
- WebEvo includes a novel technique that analyzes historic web pages to detect content-based changes.
- WebEvo includes a novel semantics-based technique that combines both text similarity and image similarity to create mappings for the changed elements in new web pages.
- We curate a valuable dataset from 13 popular web pages with the ground truth of changed elements [39], which can help the community to replicate our research and support future research.
- We conduct an empirical evaluation of WebEvo on the 13 popular websites in identifying changed elements, demonstrating the superiority over the state-of-the-art techniques, DOM-based detection and Vista, in terms of detection accuracy and runtime performance.

2 MOTIVATING EXAMPLE

In this section, we present several examples that motivate the design of WebEvo to detect semantic structure changes, filter content-based changes, and identify mappings in new web pages using semantics-based visual search.

DOM-based Evolution. Browsers adopt the HTML Document Object Model (DOM) to organize and render the Graphic User Interfaces (GUIs) of web pages. DOM is based on XML, which represent GUI objects (e.g., buttons, hyper links, and text labels) as XML elements and the relationships among these elements as a tree (i.e., the hierarchy of XML elements). Based on the DOM hierarchy, XPath is used to find the location of any element on a webpage. As web application evolves, the changes of the GUIs will be reflected by the changes in DOM elements and the added/deleted/updated web elements can be identified by analyzing the DOM tree of the web pages. For example, in Figure 1, the old web page of *www.w3schools.com* at the top contains three hyper links in the banner, where the last link is "EXAMPLES" and its XPath locator is `/html/body/div[4]/div/a[3]`; the new web page at the bottom adds a new hyper link called "EXERCISES", and its XPath locator is `/html/body/div[4]/div/a[4]`. Consider another example web page of *www.imdb.com* shown in Figure 2. As shown in the highlighted areas, the new web page contains more social links, which use more images to represent the new links. While more links are used, the XPath for the web element that holds these links remains unchanged, being `/html/body/div[3]/div/div[1]/div[2]/div[2]/ul/li[4]`.

Semantic Structure Changes. In web pages, many web elements are updated constantly due to different purposes such as weather, stock market indicator, news and advertisements. These changes usually are simple ***content-based changes***, such as news headlines, articles and images being updated, which preserve the semantic structures of the web pages and thus will not cause problems for RPA tools or test scripts. On the other hand, there are changes that can alter the semantics expressed by certain parts of a web page. We refer to this type of changes as ***semantic structure changes***.

Figure 4(a) shows the yahoo finance website from March, 2013, and Figure 4(b) shows the same web page from a few days later, where the contents - headlines, images, exchange rates - have changed but the semantic structure of the web page has remained the same. The content-based changes are highlighted within green rectangles. Finally, Figure 4(c) shows the evolved web page from November, 2013, with the currency ticker moving to the top of the page and being displayed horizontally (shown within the red rectangles), changing the semantics expressed by that part of the web page. Due to this change, a tool monitoring stock prices or exchange rates from the web page may no longer function as intended. Therefore, it is important to detect web element changes that change the semantic structure of a web page.

Mapping Elements in New Web Pages. Instead of DOM-based updates, the evolution of web elements may cause the locators of some elements used by RPA tools to be changed. As the DOM structures of these elements can be very different in new web pages, existing techniques [41] adapt CV techniques to detect similar images for detecting such changes. However, considering only images may not be sufficient. For example, in Figure 3, the "LEARN

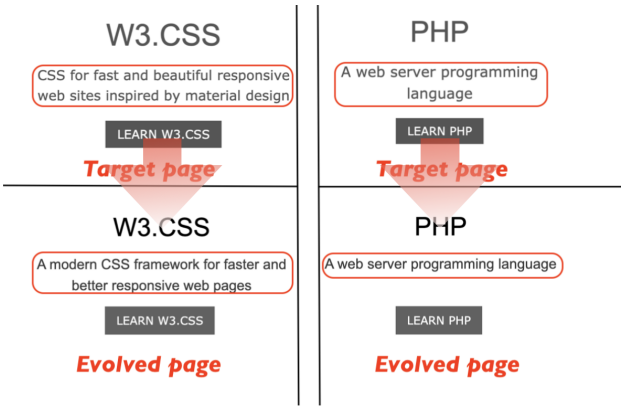
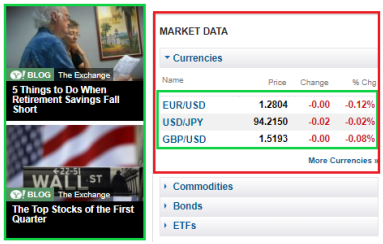
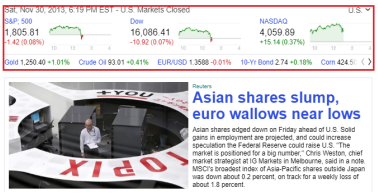


Figure 3: Mapping elements to the evolved page of *www.w3schools.com*

(a) Web page showing news headlines and a real-time stock price / currency ticker



(b) Same web page showing changed content (green rectangles) but same overall structure



(c) Evolved web page showing semantic structure changes (red rectangle)

Figure 4: Content-based and semantic structure changes on a website

W3.CSS" section in *www.w3cschools.com* has a new text content, and the "LEARN PHP" section changes the display of the text content from two lines to one line. If only image similarity is used to detect these changes, the changes will be considered as two new web elements are added, instead of one updated element (i.e., the "LEARN W3.CSS" section), resulting in inaccurate detection. This inspired us to consider a synergy approach that first compares the text similarity followed by image similarity comparison, which could significantly improve the accuracy of finding the mappings for the changed web elements.

3 DESIGN OF WEBEVO

3.1 Overview

We illustrate the overview of WebEvo in Figure 5. Our approach consists of two main modules, namely, the *Semantic Structure Change Detection* module and the *Semantic Visual Search* module. In the first module, using old and new versions of a web page as inputs, we detect all changed web elements, then filter out content-based changes and output *semantic structure changes*. This module first performs *DOM-tree based Change Detection* by comparing the DOM trees of two pages to find content-based changes and structural changes. Then the detected changes are further pruned via our *History-based Semantic Structure Change Detection* technique to output only semantic structure changes. Finally, these detected changes are used as input to our semantics-based visual search module, which performs element-wise semantics comparison through both text and image similarities to identify whether any elements in the new web pages can be mapped to the changed elements in the old web pages. Based on these detected changed elements, WebEvo provides the suggestions for making appropriate code modification for IR and RPA tools.

3.2 Semantic Structure Change Detection

Our goal is to find semantic structure changes occurring between different versions of a web page as shown in Figure 4. WebEvo performs a two-step detection: *DOM-tree Based Change Detection* and *History-based Semantic Structure Change Detection*. The first step identifies corresponding subtrees between two versions of a web page to detect content-based changes and DOM-structure changes by comparing web element attributes. To filter out content-based changes, the second step utilizes a novel history-based technique to prune the content-based changes from the output of the previous step to preserve only *semantic structure changes*.

According to the functions of web pages, we divide them into two categories: *content display* category and *content submission* category. For both types of web pages, we extract the DOM tree of a old and new versions of a web page using the HTMLCleaner tool. The two DOM trees and the part to be detected in the form of XPath are given to the DOM-tree based change detection module as input.

3.2.1 DOM-tree Based Change detection. This module detects whether a part of the web page has changed by comparing the attributes and the structure of the corresponding DOM-trees. Given two DOM trees T and T', to determine how the structure of T has changed with respect to T', WebEvo adapts the idea of *Levenshtein Edit Distance* [37] to identify the minimum element changes that can convert T to T', where an element change can be adding, deleting, or updating a node. The definitions for adding and deleting changes are straightforward: When an element e in T cannot be found in

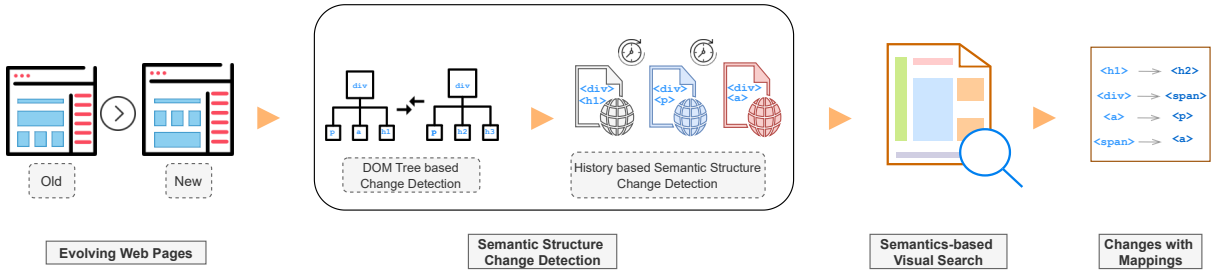


Figure 5: Overview of WebEvo

Algorithm 1: DOM-Tree Comparison

Input: DOM-Tree t, DOM-Tree t'
Output: Changed Elements C_d

1 $e = t.get("/body"), e' = t'.get("/body"), C_d = [];$
2 $TreeDiff(e, e', C_d);$
3 **return** C_d;
4 **Function** $TreeDiff(Nodee, Nodee', C_d)$
5 $map = EditDist(e.children, e'.children);$
6 **for** $pair \in map.updates$ **do**
7 $C_d.add(pair, 'update');$// identified updated changes
8 $TreeDiff(pair[0], pair[1], C_d);$// recursive checks
9 $C_d.add(map.adds, 'add'), c.add(map.deletes, 'delete');$

T', e is considered as a deleting change; when an element e' in T' cannot be found in T, e' is considered as an adding change. Detecting update changes requires more checks on the attributes, since an updated element that has the same ID attributes (i.e., `id`) and is dissimilar to the element in the old web page (i.e., preserving only a few attributes) usually represents new semantics, very likely to causes RPA tools or test scripts to fail. Based on the empirical observations, when an element is updated in the new web page, its ID attributes (i.e., *id*, *class*, and *tag*) are often preserved[1], and most of its child nodes are still the same. Thus, WebEvo considers an element e as a candidate for an update change only if it can find another element that shares e's ID attributes and have similar child nodes; otherwise, e is considered a deleted node that cannot be found in the new web page.

Algorithm 1 shows how WebEvo compares two DOM trees T and T'. WebEvo first extracts the **body** elements (i.e., e and e') from both the trees, and uses the recursive function $TreeDiff$ to compute the changes for the subtrees rooted at e and e' (Lines 1-3). $TreeDiff$ computes the minimum changes that can convert the child elements of e to the child elements of e' using the Levenshtein Edit Distance. Here, a child node of e is considered to be identical with a child node of e' if one of three ID attributes (id, class, and tag) of e and e' have identical values. By applying the edit distance algorithm based on this change definition, WebEvo identifies the matched nodes from the child nodes of e to the child nodes e'. For each pair (n, n') of the matched nodes, if (1) n and n' has only one ID attribute that has the same values and (2) the tag names of n's child nodes and n''s child nodes have more than 30% different values, then n and n' are considered not similar enough, and a corresponding adding or a deleting change is reported. Otherwise, an update change is reported for the attributes of n and n' if any (Line 7), and a recursive call of $TreeDiff$ is invoked to identify changes of the child nodes of n and n' (Line 8). For certain child nodes of e, if WebEvo fails to find the matched nodes from e''s child nodes, WebEvo reports these child nodes of e as adding changes and deleting changes correspondingly (Line 9).

3.2.2 History-Based Semantic Structure Change Detection. Due to the constantly evolving dynamic web content, the DOM-tree based change detection alone cannot differentiate between content-based changes and semantic structure changes, and its detection results C_d contains both content-based and semantic structural changes. Thus, we design the history-based semantic structure change to detect the content-based changes (C_c) and output only semantic structure changes ($C_d - C_c$).

Differentiating content-based changes from semantic structure changes is a challenging task. The indicators of semantic structure are different from one web page to another. For example, in Figure 4(a), both the Strings "Name" and "+0.10%" are text elements in DOM. But the change from "Name" to "Code" is regarded as a semantic structure change of the web page (because the type of information under "Name" is different from the type of information under "Price"), while the change from "+0.10%" to "-0.02%" is considered as a content-based change.

To address this challenge, we leverage an important insight that the history of the web page can reflect which elements of the web page present the structural information. Specifically, the structural elements of a web page usually remain unchanged across multiple versions of the web page within a relatively short time span (e.g., one or two days). For example, Figure 4(a) and Figure 4(b) show two versions of a web page. The structural elements such as "Currencies" and "Name" remain unchanged while content elements such as currency exchange rates change frequently. Based on this information, this module is able to detect the elements whose contents (i.e., text and images) change constantly across historic pages.

Algorithm 2 shows the details on detecting content-based changes using historic web pages. The algorithm receives the DOM tree of a web page t and the DOM trees of its historic web pages H as input, and compares t with each DOM tree h from H. The algorithm starts the comparison from the body elements of t and h (Lines 1-3). It invokes the function $findMatchedNodes$ to find all the matched nodes from t and h (Line 4): for each pair of matched nodes, the two

[1]In our evaluation dataset, ~ 98% of the updated elements preserve the ID attributes.

Algorithm 2: Detecting Content-based Changes

Input: DOM-Tree t, Historic DOM-Trees H
Output: Content-based Changes C_c

```
1 e = t.get("/body");
2 for h ∈ H do
3     e' = h.get("/body");
4     P, U = findMatchedNodes(e, e');
5     C_c.addAll(U);// record unmatched nodes
6     for p ∈ P do
7         findChanged(p[0], p[1], C_c); // recursive search
8 return C_c;
```

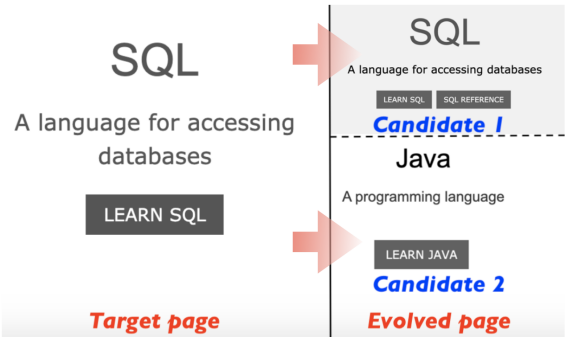


Figure 6: Candidate elements added to max heap

subtrees rooted at the matched nodes must have the same structure and all nodes should have the same tag names. For the unmatched nodes in t, they are considered as content-based changes (Line 5). For each pair of matched nodes, the algorithm invokes the function *findChanged* to check whether the matched nodes have the same texts and the same image urls; if not, a content-based change is reported (Line 7). Note that *findChanged* performs the search recursively on the child nodes of each matched nodes as well.

Finally, WebEvo combines the information from the steps of text analysis and graphical element analysis as features to map the old web page and the new web page. Using the features, WebEvo computes the similarity scores between two elements of a web page. Based on the similarity scores, the elements having the highest similarity are chosen for internal structure analysis to identify content-based changes.

3.3 Semantics-based Visual Search

As illustrated in Section 2, changed elements may update their locations in the web pages but preserve their semantics and appearance. If these elements are not correctly identified, they will be considered as deleted elements in the old web page and new elements in the new web page, losing the opportunities to fix IR and RPA tools. To address this problem, WebEvo computes both text similarity and image similarity of the elements in both the old and the new web pages to identify such changed elements.

Semantics-Based Element Mapping. Algorithm 3 shows the algorithm for semantics-based visual search. The algorithm accepts as input, C_s, the output from the semantic structure change detection module (i.e., $C_s = C_c - C_d$). For each detected changed element

Algorithm 3: Semantics-based Element Mapping

Input: Target Element e, Candidate Elements C_s
Output: Matched Element e'

```
1  if e has text then
2      C'_s = max heap;
3      for Candidate Element e' ∈ C_s do
4          e'.textSim = compareText(e.text, e'.text) ;
5          C'_s.add(e'); // sort based on e'.textSim
6      while C'_s is not empty do
7          e' = C'_s.pop() ;
8          sim = compareImage(e, e') ;
9          if sim == true then
10             return e';
11 else
12     for Candidate Element e' ∈ C_s do
13         sim = compareImage(e, e') ;
14         if sim == true then
15             return e';
```

e, the algorithm aims to find an element in the candidate elements C_s in the new web page that has similar content as e's content. The candidate elements are obtained from all the elements in the new web page, such as `<div>`, `<h1>` and `<p>`. As a web element may or may not have text content (Line 1), the algorithm first checks whether the element has text content, and directly computes image similarity for the elements without text content (Lines 11-15). Next, the algorithm computes the text similarity using *Levenshtein Edit Distance* [37] for each candidate element and sort them using a max heap (Lines 3-5). While synonyms based on Wordnet [19] may be used, they are less effective when an element's text contains phrases or sentences, while Levenshtein distance is effective in detecting simple word updates and even sentence updates. The text similarity ensures that the algorithm prioritizes the elements with higher text similarities when comparing image similarity. For example, in Figure 6, the target web element is on the left, which is the "LEARN SQL" section, and two candidate elements are shown on the right. The text similarity between the target element and the "Candidate 1" is 0.77, and the similarity for the "Candidate 2" is 0.33. Thus, "Candidate 1" will have a higher priority for image comparison. Finally, the algorithm pops the candidate elements from the max heap, and computes image similarity to detect matching element that has high content similarity (Lines 6-10).

Image Similarity of Element Screenshots. The image similarity between elements can be influenced by many factors, such as size, background color and other elements in the web pages. Existing techniques, such as Vista [41], takes the screenshot of the entire web page and applies template matching [42] for finding similar elements. However, such techniques are easily affected by background colors and the images of other elements that are close to the candidate elements. To address this problem, our algorithm takes the screenshots of the elements for computing image similarity, rather than using template matching to search the screenshot of the whole web page. Algorithm 4 shows the detailed steps of the image similarity computation.

Table 1: Details of the websites used in the evaluations

Website	Category	# Elements	LOC	Collected Date		
				History Pages	Target Page	Evolved Page
www.w3schools.com	Education	571	814	01/11/2016-01/13/2016	01/14/2016	11/01/2019
www.foodnetwork.com	Food and Drink	1,465	5,657	10/27/2018-10/29/2018	11/01/2018	11/04/2019
music.douban.com	Arts and Entertainment	1,925	2,299	02/11/2014-03/09/2014	02/08/2014	02/07/2019
beijing.douban.com	Travel	1,509	2,353	10/10/2017-10/25/2017	10/03/2017	10/07/2019
book.douban.com	Education	2,622	3,691	08/26/2016-08/29/2016	08/19/2016	08/05/2019
movie.douban.com	Arts and Entertainment	1,583	1,799	10/07/2014-10/22/2014	11/15/2014	11/08/2019
www.amazon.com	E commerce and Shopping	2,796	1,802	08/01/2017-08/03/2017	08/04/2017	08/06/2020
www.apple.com	Computers Electronics and Technology	535	581	02/06/2018-02/08/2018	02/09/2018	08/07/2020
www.classdojo.com	Education	615	126	06/01/2017-06/03/2017	06/05/2017	07/22/2020
www.homedepot.com	Business	910	2,173	07/12/2019-07/14/2019	07/15/2019	08/04/2020
www.linkedin.com	Community and Society	622	43	08/12/2019-08/14/2019	08/15/2019	08/17/2020
www.usps.com	Community and Society	817	2,586	08/05/2018-08/07/2018	08/08/2018	08/08/2020
www.xfinity.com	Internet	771	1,405	08/01/2018-08/03/2018	08/04/2018	08/04/2020
Total		16,741	25,329	-	-	-

Algorithm 4: Image Similarity

Input: Element e, Element e'
Output: true/false

```
1  original = getPicSim(e, e'); ;
2  if original >= threshold_i then
3  |  return true; // return if original images can match
4  else
5  |  greyScale(e); // convert e to greyscale
6  |  greyScale(e'); // convert e' to greyscale
7  |  grey = getPicSim(e, e');
8  |  if grey >= threshold_i then
9  |  |  return true; // if images can match after greyscale
10 |  else
11 |  |  invertColor(e'); // invert the colors of e'
12 |  |  if getPicSim(e, e') >= threshold_i then
13 |  |  |  return true; // images can match after inversion
14 return false;// cases above all fails
```

Given two elements e and e', the image similarity is mainly computed using image hashing techniques that compute hash values for the pixels, convert the hash values into bit sequences, and compare the similarity of the bit sequences [23, 27] (Line 1). If the similarity is above the threshold $threshold_i$, then a matching element is found (Lines 2-3). If not, then the algorithm applies image mutation and recompute the similarities (Lines 4-14). The reason is that images in web pages may involve updates on colors, WebEvo adopts existing image mutation techniques, i.e., grey scale conversion (Lines 5-10) and color inversion [42] (Lines 11-13), to mitigate the color evolution problem. For some elements, grey scale conversion may not be effective to improve similarity computation. As these elements use black and grey colors, grey scale conversion will not help. To address this problem, WebEvo applies color inversion (Line 11) to invert the colors of the element and compute the similarity on the mutated image.

4 EVALUATION

We implement WebEvo upon Selenium [6] to locate web elements and take screenshots of web elements, and upon OpenCV [42] to mutate the collected screenshots and compute image similarity. The versions of the Google Chrome browser and the Chrome Driver [9] used for the semantics-based visual search module are 90.0.4430.93 and 90.0.4430.24, respectively. The evaluations are conducted on a Macbook Pro with a Dual-Core Intel Core i5 processor (2.3GHz) and 16 GB RAM.

We evaluate WebEvo on the web pages of 13 real-world web applications for identifying changed elements. Specifically, we aim to answer the following research questions:

- *RQ1: How effective is WebEvo in identifying changed elements in web pages, compared to the state-of-the-art visual web repair approach, Vista [41]?*
- *RQ2: How effective can the changed content identification and semantics-based visual search be in improving WebEvo's effectiveness?*
- *RQ3: What is the runtime performance of WebEvo, compared to Vista?*
- *RQ4: How effective is WebEvo in detecting changed elements that affect RPA and IR tools?*

4.1 Subjects and Evaluation Setup

We collect popular web pages from the representative web applications in most popular categories [2–4] (e.g., e-commerce, entertainment, business, and job) as our evaluation subjects. These websites provide daily services to meet various needs of users and have great values for RPA tools to automate the process in these websites. For each chosen web applications, we aim to collect a target page, an evolved page, and three historic pages to detect changed elements. We downloaded these web pages from Wayback Machine [7], which archives the historic web pages of popular websites. Note that we included only the complete web pages in our evaluations, as incomplete web pages with missing images will introduce noises to the visual search module. To ensure that the collected web pages will have substantial amount of changed elements for evaluations, we select web applications to collect web pages based on two rules.

Table 2: Evaluation results of WebEvo and Vista

Website	# Changed Elements				WebEvo				Vista *			
	Add	Delete	Update	Total	Found	Prec.	Rec.	F_1	Found	Prec.	Rec.	F_1
www.w3schools.com	68	20	84	172	147	0.96	0.82	0.88	87	0.37	0.31	0.34
www.foodnetwork.com	8	5	19	32	26	0.92	0.75	0.83	21	0.19	0.17	0.18
music.douban.com	40	18	12	70	58	0.95	0.79	0.86	23	0.43	0.33	0.37
beijing.douban.com	5	6	16	27	22	0.82	0.67	0.74	13	0.85	0.50	0.63
book.douban.com	15	12	14	41	35	0.91	0.78	0.84	19	0.53	0.38	0.44
movie.douban.com	22	10	16	48	43	0.95	0.85	0.90	16	0.81	0.50	0.62
www.amazon.com	71	19	39	129	97	0.95	0.71	0.81	25	0.32	0.14	0.19
www.apple.com	14	5	23	42	37	0.95	0.83	0.89	22	0.27	0.21	0.24
www.classdojo.com	4	11	12	27	20	0.90	0.67	0.77	20	0.55	0.48	0.51
www.homedepot.com	25	25	13	63	57	0.93	0.84	0.88	25	0.84	0.55	0.66
www.linkedin.com	16	9	25	50	51	0.88	0.86	0.87	30	0.53	0.47	0.50
www.usps.com	33	28	19	80	81	0.83	0.84	0.83	40	0.38	0.32	0.35
www.xfinity.com	18	11	22	51	45	0.91	0.80	0.85	26	0.08	0.06	0.07
Average	26	14	24	64	55	0.91	0.79	0.84	28	0.47	0.34	0.39

* Results of Vista consider only deleted and updated elements since Vista cannot handle added elements.

First, the web applications should be popular and their web pages should contain a large number of elements. This will rule out simple web pages with a few web elements such as *www.google.com*. Second, for the web pages on different dates, there should be at least 20 changed elements that belong to different element types (e.g., **<h>** and ** **). As redesigning a whole web site requires users to spend more efforts in using the new web site, web sites do not tend to revamp the whole web site within a short period, but adopt a lightweight and progressive approach to upgrade the web sites [5, 8]. Thus, for each web site, we choose a target web page at a date and an evolved web page at a later date, and the average differences between these two dates are more than 2 years to ensure that there are enough changed elements between the target web page and the evolved web page.

In total, we obtain 13 websites whose changed elements are above 20. For each web page, we used Chrome browser's extension GoFullPage [10] to capture the screenshot of the whole page, which can be used for changed element detection. Table 1 shows the details of the web applications and the collected web pages for the real-world dataset. Column "Website" shows the web sites from which we collect the web pages and Column "Category" shows which popular categories the web sites belong to. Column "# Elements" shows the number of web elements (e.g., **<h>** and ** **) for a page and Column "LOC" shows the lines of HTML code in the page. Column "Collected Date" shows the dates for the collected history pages, target pages, and the evolved pages.

Ground Truth of The Changed Elements. For each web page of the collected web application, we perform a two-step inspection: ⓐ we first applied the DOM-tree based detection of WebEvo on both the target weg pages and the evolved web pages to identify the changed elements, and then manually inspected both the DOM trees of the target and the evolved web pages using the browser's inspector tool to confirm whether the changed elements are indeed changed in the new web page. We excluded the popped up elements as Selenium cannot capture their screenshots, and thus they cannot be analyzed by the visual search module. ⓑ as the DOM-tree based detection may miss some changed elements, we further render the target and the evolved web pages and manually inspect each element of the target web page to confirm whether they are changed in the new web page. By combining the results of ⓐ and ⓑ, we can obtain the ground truth of the changed elements for each collected web application.

Evaluation Metrics. To measure the effectiveness of WebEvo, we compute the precision, recall, and F_1 values for the detected changed elements. A changed element can be a new element, a deleted element, or an updated element. For each reported detected changed element, we manually compare it with the ground truth to confirm whether it is a true positive (TP); otherwise, it is considered as a false positive (FP). In particular, for a changed element, we also verify the detected mapping; if the mapping is incorrect, we will consider it as a FP. If an element does not change and WebEvo does not report it as an changed element, then it is considered as a true negative (TN); otherwise, if an element changes and WebEvo misses it, it is considered as false negative (FN). Based on these values, we compute the precision using $\frac{TP}{TP+FP}$, the recall using $\frac{TP}{TP+FN}$, and the F_1 score using $2 * \frac{prec*rec}{prec+rec}$.

4.2 RQ1 Overall Effectiveness

We compared WebEvo with DOM-tree based detection, and Vista [41], a state-of-the-art visual web repair approach that automatically fixes web tests by finding changed elements. We implemented a DOM-tree based detection tool based on the algorithm described in Section 3.2.1. This detection tool adopts the same idea as the existing work [18, 22], which compares the attributes and the structures of DOM trees to detect changed elements. As Vista provides a released tool, we directly used it for comparison. We applied the DOM-tree based detection tool, Vista, and WebEvo on the 13 websites and compare their effectiveness. We compare WebEvo with the DOM-tree based detection tool in detecting the added, detected, and updated elements. Note that Vista receives a locator (i.e., an XPath) in the target web page as input and cannot effectively detect new elements in the evolved page, since the locators of the new elements only exist in the evolved web page. Also, Vista cannot detect

Figure 7: Top navigation bar in the changed page

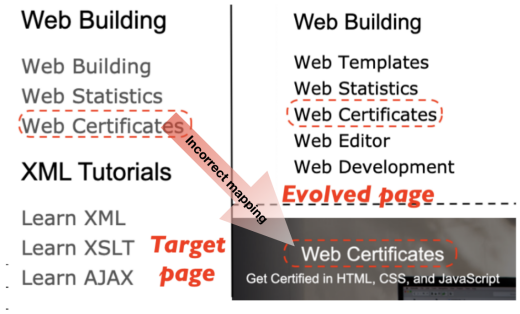


Figure 8: Incorrect mapping for the "Web Certificate" element

content-based changes as it lacks the capabilities to analyze historic web pages. For fair comparison, we compare WebEvo with Vista in detecting only the deleted and the updated elements without the detected content-based changes.

Comparison with DOM-based Detection. Table 3 shows the comparison results of WebEvo and the DOM-tree based detection tool, and Table 2 shows the comparison results of WebEvo and Vista. Overall, WebEvo effectively identifies changed elements in the websites with high precision (0.91), recall (0.79), and F_1 values (0.84), while the DOM-tree based detection tool achieves low precision (0.40), recall (0.73), and F_1 values (0.50). That is, WebEvo achieves a significant improvement (68.0% for F_1 score) over the DOM-tree based detection tool. These results demonstrate that by combining DOM-tree based detection and semantics-based visual search, WebEvo effectively identifies not only new or deleted elements, but also updated elements that have property or location changes.

Comparison with Vista. Compared with Vista, we can see that WebEvo is able to achieve 0.91 precision while Vista achieves only a 0.47 precision on the deleted and updated elements. On average, Vista achieves a F_1 score of 0.39, which is significantly worse than WebEvo (0.84). The main reason is that Vista considers only the screenshots when doing visual search, while WebEvo considers both text content and image content. For example, Figure 7 shows the top navigation bar of "w3cschool", where the appearances of texts in the navigator bar are changed in the new web page. As all the menu items in the navigator bar share the similar looks, Vista fails to find any matched element. When WebEvo is applied on this example, WebEvo considers both text and image similarities, and is able to correctly identify the changed element (i.e., correctly mapping the "TUTORIALS" menu items in the new web page).

FPs and FNs. Next, we describe scenarios where WebEvo produces false positives and false negatives. The main reason for the false positives is that when there are multiple similar web elements in the evolved page, WebEvo may identify an incorrect mapping for the target element. Figure 8 shows an example evolved web page that contains two similar "Web Certificate" elements. When WebEvo is applied to detect the mapping for the "Web Certificate" element in the target page, it incorrectly identifies the "Web Certificate" element at the bottom due to their same text contents and similar appearances. The false negatives are mainly caused by the changed elements that have no text content and changed the appearance substantially. For example, in Figure 1, the logo of "w3school" is an image web element without text property in the target page. However, the logo in the evolved web page has been changed to a text element with a new CSS style. Since the "w3school" element has no text content, WebEvo directly computes the image similarity and fails to identify the changed element. Similarly, Vista also fails to detect such mapping due to the appearance change of the element.

4.3 RQ2 WebEvo's Novel Techniques

To evaluate the effectiveness of changed element detection and semantics-based visual search, we compare the detected changed elements for the web pages with and without using the semantics-based visual search. The results are shown in Table 3. Column "# Changed Elements" shows the number of changed element for each target page of the web applications. Column "WebEvo w/o visual search" shows the precision, recall, and F_1 values without using semantics-based visual search. As we can see, the precision, recall, and F_1 values drop to 0.58, 0.72, and 0.64 without using semantics-based visual search. That is, semantics-based visual search achieves 31.3% improvement for the F_1 score, which is a substantial improvement. Also, on average there are 40 detected content-based changes for each web page, which are related to weather, advertisement, news, and so on. Without filtering these content-based changes, the precision of the detected changed elements will further drop to 0.40 from 0.58, which demonstrates the effectiveness of detecting content-based changes.

4.4 RQ3 Runtime Performance

We measure the runtime performance of WebEvo and compare with Vista. As DOM-tree based analysis and history-based semantic structure change detection require only a few seconds to run, we mainly compare the runtime performance of both tools' visual analysis. Table 4 shows the average analysis time for each changed element in our evaluations. On average, to finish the analysis, WebEvo requires 42.01% less time than Vista (37.04*s* v.s. 63.92*s*). Also, WebEvo is more efficient than Vista in every website. This shows that applying CV techniques to compare whole web pages require significantly more time than element-by-element comparison.

4.5 RQ4 Detecting RPA and IR Breakages

We evaluate the effectiveness WebEvo to detect web change elements that can prevent RPA and IR bots from executing their tasks.

Table 3: Effectiveness results of WebEvo's techniques

Website	DOM-based Detection				# Content-based Changes	WebEvo w/o visual search			
	Found	Prec.	Rec.	F_1		Found	Prec.	Rec.	F_1
www.w3schools.com	229	0.46	0.78	0.58	24	205	0.52	0.78	0.62
www.foodnetwork.com	162	0.12	0.71	0.21	135	27	0.74	0.71	0.72
music.douban.com	106	0.45	0.76	0.57	45	61	0.79	0.76	0.77
beijing.douban.com	119	0.06	0.44	0.11	96	23	0.30	0.44	0.36
book.douban.com	72	0.39	0.76	0.52	29	43	0.65	0.76	0.70
movie.douban.com	165	0.19	0.82	0.31	111	54	0.56	0.81	0.66
www.amazon.com	149	0.60	0.71	0.65	29	120	0.75	0.71	0.73
www.apple.com	58	0.36	0.75	0.49	11	47	0.43	0.74	0.54
www.classdojo.com	21	0.43	0.50	0.46	0	21	0.43	0.50	0.46
www.homedepot.com	68	0.62	0.81	0.70	6	62	0.68	0.81	0.74
www.linkedin.com	68	0.37	0.78	0.50	4	64	0.39	0.78	0.52
www.usps.com	97	0.58	0.81	0.68	12	85	0.64	0.81	0.72
www.xfinity.com	73	0.56	0.80	0.66	17	56	0.68	0.79	0.73
Average	107	0.40	0.73	0.50	40	67	0.58	0.72	0.64

Table 4: Runtime Performance (seconds)

Web App	WebEvo	Vista
www.w3schools.com	17.40	67.94
www.foodnetwork.com	52.50	52.90
music.douban.com	33.46	35.87
beijing.douban.com	63.15	107.95
book.douban.com	96.36	122.50
movie.douban.com	71.18	72.62
www.amazon.com	55.75	73.87
www.apple.com	12.77	46.02
www.classdojo.com	16.80	77.71
www.homedepot.com	20.00	43.47
www.linkedin.com	12.17	43.69
www.usps.com	13.95	62.59
www.xfinity.com	16.07	23.80
Average	37.04	63.92

For different types of elements, we base our evaluation on determining the percentage of changed elements that are detected. Table 5 lists 11 websites where we found element changes through manual inspection that could break RPA and IR tools. For each website, we noted how many element changes were detected by WebEvo for each element type that were changed, categorized by breakage type. Combining all the changed elements across all the websites with RPA breakages, WebEvo detected 65% of the changes. For IR breakages, WebEvo detected 100% of the changes. Most undetected changes can be attributed to WebEvo's detection not being granular enough. One such example is shown in Fig 9, where two `<a>` elements (within red rectangles) have changed, but instead of detecting those two changes, WebEvo reports the container `<div>` (green rectangle) element as changed.

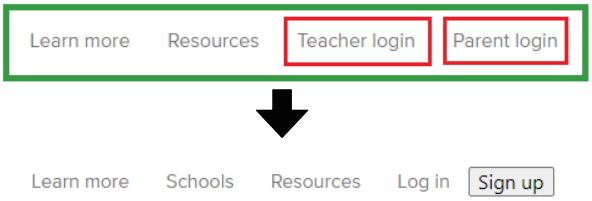


Figure 9: Change Detection that is not granular enough

5 DISCUSSION

Content-based Changes. WebEvo detects content-based changes by checking the texts and images across historic web pages. Our evaluation shows the effectiveness of the detection and how it reduces the false positives of WebEvo. However, it is possible that some websites rename image files, give the same file names to different images, or use same texts for a couple of days in the content retrieved from web servers, which will cause WebEvo to produce false positives and false negatives. Even though we mitigate such issue by collecting historic web pages within a period, this issue can be further addressed by employing CV techniques to further compare the image similarity and checking more attributes when the text remains the same. Alternatively, if we can obtain the server code, which is not the assumption for this work, we may perform static analysis to identify elements whose contents are constantly changed based on the server output.

Semantics-based Visual Search. Our semantics-based visual search adopts the string similarity based on the Levenshtein distance and the image similarity based on fingerprint to identify the mappings for changed elements in new web pages. While the evaluations show promising results for this approach, the string similarity may miss synonyms that have low string similarity but high semantic similarity. This may be improved by combining string similarity with semantic similarity such as sentence similarity [11, 26] or semantic patterns [36, 40, 48]. WebEvo computes image similarity by hashing images into bit sequences and computing the similarities of the bit sequences [23, 27]. This may not always produce desirable results due to the limitations of the hashing functions and may be improved by using object recognition such as SIFT/FAST [28, 29, 38]. Additionally, the synergy of string similarity and image similarity

Table 5: Performance of WebEvo in detecting RPA and IR breakages

Website	Breakage Type	Changed Element(s)	Number of Changes (Per Element)	Number of Detected Changes (Per Element)
www.amazon.com	RPA	<a>	4	2
www.classdojo.com	RPA	/ <a>	1 / 2	1 / 0
www.xfinity.com	RPA		4	4
www.usps.com	RPA	/ <a> /	3 / 1 / 1	3 / 1 / 1
music.douban.com	RPA		6	2
www.homedepot.com	RPA		1	1
www.foodnetwork.com	RPA		4	2
beijing.douban.com	RPA	<a>	6	3
www.linkedin.com	RPA	<section>/	1 / 3	1 / 2
movie.douban.com	IR		4	4
book.douban.com	IR		1	1

adopted by WebEvo may be replaced with deep learning techniques that model the joint semantics of text and images [14, 21, 25, 47].

Threats to Validity. The main internal threat comes from the mistakes we may make during the labelling of the changed elements for each website used in our evaluations. To reduce the threat, each changed element was verified by at least two authors. We further checked the HTML code, and when we were not sure which changed elements caused the UI changes we used tools (e.g., the browser inspector tool of Google Chrome) to identify the corresponding UI parts for the changed elements. There are two main external threats to validity. First, we recognized the need to tailor WebEvo to different types of web pages to obtain effective outcomes. Thus, we choose the representative websites in most popular categories and make sure that the old and the new web pages have non-trivial changes in terms of changed elements. Second, WebEvo performs comparison on the pre-defined attributes or tags to detect changed elements, which works well given that HTML rarely introduces new types of elements. WebEvo can be easily extended to support more types of web elements by expanding the pre-defined lists of the attributes and tags.

6 RELATED WORK

Web Test Repair. WATER [18] runs a web test, collects the DOM properties about the test breakages, and computes the similarity of the DOM properties for web elements to repair the tests. Built upon WATER, WATERFALL [22] repairs test breakages in the intermediate minor versions between two major releases of a web application. By using CV techniques, Vista [41] achieves better performance than WATER in repairing web tests. While WebEvo also uses CV techniques, WebEvo uses a finer grained visual information (i.e., the screenshot of an element versus the screenshot of the whole web page) and text semantics to improve the visual search of web elements, and our evaluations have shown the superiority of WebEvo over Vista.

Web Application Repair. Besides changes due to web application evolution, there are other unexpected changes of web elements that may cause problems. Mahajan et al. [31] propose automated repair techniques to address presentation changes when using different browsers. Their later work [12, 30, 32] also propose repair techniques to address mobile-friendly problems and internationalization problems in web applications. Cassius [35] and its extension VizAssert [34] provide an extensible framework for reasoning about web pages' layout and can be used to repair faulty CSS in web applications. While these repair techniques are related to WebEvo in terms of detecting inconsistent web elements across different web pages, WebEvo focuses on detecting changed elements (i.e., new, deleted, and updated elements) and these techniques focus on the same web elements that have unexpected presentations under different settings.

CV Techniques for Software Engineering Tasks. REMAUI [33] adapts CV techniques for reverse engineering UIs of mobile apps. Sikuli [15, 50] identifies and controls UI components for automating UI testing via image recognition. WebDiff [17] and XPERT [16] identify visual differences using CV techniques, aim to detect cross browser rendering issues. IconIntent [49] collects a set of icons that represent sensitive data and uses CV techniques to determine whether the icons used by UI widgets are similar to the collected sensitive icons in Android apps. Besides CV techniques, WebEvo analyses web pages collected at different timestamps to identify changed contents, and further applies text similarity to choose candidate web element for finding the mappings across different versions of the web pages.

DOM Tree-based Web App Analysis. The DOM structure-based methods typically parse the target and the evolved web page into a DOM tree and then uses an algorithm to calculate the similarity between the trees to find the subtree in the evolved version of the DOM tree that is closest to the target subtree structure of the target version. Flesca et. al. proposed a subtree similarity comparison approach using a bipartite graph based maximum matching algorithm [20]. A node signature comparison algorithm to find textual changes in the DOM trees of the target and evolved web pages is presented in [24]. However, the general complexity of the tree similarity calculation method is relatively high and the overall performance is not high under the trend of increasingly complicated web pages. The results obtained based on only DOM information are also not accurate.

Machine Learning-based Web App Analysis. Machine learning based approaches extract the characteristics of web pages by analyzing a large number of web pages to classify various web pages.

Each category of web pages is considered to have a similar data extraction method. Through this machine learning method, when the page changes, only a new analysis of the new version of the web page is required to automatically generate a new data extraction tool. Borgolte et. al. presented an approach combining DOM-based comparison and clustering techniques to detect relevant changes on web pages [13]. However, machine learning approaches requires the support of a large amount of data and has high requirements on the computational power of the device.

7 CONCLUSION

We have presented a novel framework, WebEvo, for monitoring web element changes that can break IR tools and web test scripts. Our tool performs DOM-based comparison between old and new versions of a web page and carries out novel non-content change detection using semantic and graphical analysis to filter out irrelevant changes and finally, a novel semantics-based visual search technique is used to refine the detected changes. We showed WebEvo's practicality by evaluating our approach on datasets constructed from popular real-world websites and demonstrating substantially better detection and runtime performances. We also showed how WebEvo can detect breakages for IR and RPA tools.

ACKNOWLEDGMENTS

Xusheng Xiao's work is partially supported by the National Science Foundation under the grants CCF-2046953 and CNS-2028748. Yanfang Ye's work is partially supported by the National Science Foundation under the grants IIS-2027127, IIS-2040144, IIS-1951504, CNS-2034470, CNS-1940859, CNS-1814825, and OAC-1940855, the DoJ/NIJ under the grant NIJ 2018-75-CX-0032. Ying Zhang's work is supported by the National Key Research and Development Program of China. Xusheng Xiao is the corresponding author.

REFERENCES

[1] 2020. IBM Robotic Process Automation as a Service with WDG Automation. https://www.ibm.com/products/robotic-process-automation.
[2] 2020. Most Popular Types of Websites. https://www.hostgator.com/blog/popular-types-websites-create/.
[3] 2020. Most Popular Types of Websites. https://www.websitebuilderexpert.com/designing-websites/types-of-websites/.
[4] 2020. Most Popular Types of Websites. http://www.webyurt.com/popular-types-of-websites.
[5] 2020. Poor Sales? Maybe You Need a Website Redesign: Here's How. https://www.crazyegg.com/blog/website-redesign-tips/.
[6] 2020. Selenium. https://www.selenium.dev/.
[7] 2020. Wayback Machine. https://archive.org/web/.
[8] 2020. Why redesigns don't make users happy. https://uxplanet.org/why-redesigns-dont-make-users-happy-f1b29cc940ce.
[9] 2021. Chrome Driver. https://chromedriver.chromium.org/downloads.
[10] 2021. GoFullPage. https://gofullpage.com/.
[11] Palakorn Achananuparp, Xiaohua Hu, and Xiajiong Shen. 2008. The evaluation of sentence similarity measures. In *Proceedings of the International Conference on Data Warehousing and Knowledge Discovery (DaWaK)*. 305–316. https://doi.org/10.1007/978-3-540-85836-2_29
[12] Abdulmajeed Alameer, Paul Chiou, and William G.J. Halfond. 2019. Efficiently Repairing Internationalization Presentation Failures by Solving Layout Constraints. In *Proceedings of the IEEE International Conference on Software Testing, Verification, and Validation (ICST)*. 172–182. https://doi.org/10.1109/ICST.2019.00026
[13] Kevin Borgolte, Christopher Kruegel, and Giovanni Vigna. 2014. Relevant change detection: a framework for the precise extraction of modified and novel web-based content as a filtering technique for analysis engines. In *Proceedings of the International Conference on World Wide Web (WWW)*. 595–598. https://doi.org/10.1145/2567948.2578039
[14] Matthew Browne and Saeed Shiry Ghidary. 2003. Convolutional Neural Networks for Image Processing: An Application in Robot Vision. (2003), 641–652. https://doi.org/10.1007/978-3-540-24581-0_55
[15] Tsung-Hsiang Chang, Tom Yeh, and Robert C. Miller. 2010. GUI testing using computer vision. In *Proceedings of the International Conference on Human Factors in Computing Systems (CHI)*. 1535–1544. https://doi.org/10.1145/1753326.1753555
[16] Shauvik Roy Choudhary, Mukul R. Prasad, and Alessandro Orso. 2014. X-PERT: a web application testing tool for cross-browser inconsistency detection. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*. 417–420. https://doi.org/10.1145/2610384.2628057
[17] Shauvik Roy Choudhary, Husayn Versee, and Alessandro Orso. 2010. WEBDIFF: Automated identification of cross-browser issues in web applications. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM)*. 1–10. https://doi.org/10.1109/ICSM.2010.5609723
[18] Shauvik Roy Choudhary, Dan Zhao, Husayn Versee, and Alessandro Orso. 2011. WATER: Web Application TEst Repair. In *Proceedings of the First International Workshop on End-to-End Test Script Engineering (ETSE)*. 24–29. https://doi.org/10.1145/2002931.2002935
[19] Christiane Fellbaum (Ed.). 1998. *WordNet An Electronic Lexical Database* . MIT Press. https://doi.org/10.1017/S0142716401221079
[20] Sergio Flesca, Filippo Furfaro, and Elio Masciari. 2001. Monitoring Web information changes. In *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC)*. 421–425. https://doi.org/10.1109/ITCC.2001.918833
[21] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. 2016. *Deep Learning*. MIT Press. https://doi.org/10.1007/s10710-017-9314-z
[22] Mouna Hammoudi, Gregg Rothermel, and Andrea Stocco. 2016. WATERFALL: an incremental approach for repairing record-replay tests of web applications. In *Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*. 751–762. https://doi.org/10.1145/2950290.2950294
[23] Wei Jiang, Guihua Er, Qionghai Dai, and Jinwei Gu. 2006. Similarity-based online feature selection in content-based image retrieval. *IEEE Transactions on Image Processing (TIP)* 15, 3 (2006), 702–712. https://doi.org/10.1109/TIP.2005.863105
[24] HP Khandagale and PP Halkarnikar. 2010. A novel approach for web page change detection system. *International Journal of Computer Theory and Engineering (IJCTE)* 2, 3 (2010), 364–368. https://doi.org/10.7763/IJCTE.2010.V2.168
[25] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444. https://doi.org/10.1038/nature14539
[26] Yuhua Li, David McLean, Zuhair A Bandar, James D O'shea, and Keeley Crockett. 2006. Sentence similarity based on semantic nets and corpus statistics. *IEEE transactions on knowledge and data engineering (TKDE)* 18, 8 (2006), 1138–1150. https://doi.org/10.1109/TKDE.2006.130
[27] Fuhui Long, Hongjiang Zhang, and David Dagan Feng. 2003. *Fundamentals of Content-Based Image Retrieval*. Springer Berlin Heidelberg, 1–26. https://doi.org/10.1007/978-3-662-05300-3_1
[28] David G. Lowe. 1999. Object Recognition from Local Scale-Invariant Features. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 1150–1157. https://doi.org/10.1109/ICCV.1999.790410
[29] David G. Lowe. 2004. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision (IJCV)* 60, 2 (2004), 91–110. https://doi.org/10.1023/B:VISI.0000029664.99615.94
[30] Sonal Mahajan, Negarsadat Abolhasani, Phil McMinn, and William G.J. Halfond. 2018. Automated Repair of Mobile Friendly Problems in Web Pages. In *Proceedings of the International Conference on Software Engineering (ICSE)*. 140–150. https://doi.org/10.1145/3180155.3180262
[31] Sonal Mahajan, Abdulmajeed Alameer, Phil McMinn, and William G.J. Halfond. 2017. Automated Repair of Layout Cross Browser Issues Using Search-Based Techniques. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*. 249–260. https://doi.org/10.1145/3092703.3092726
[32] Sonal Mahajan, Abdulmajeed Alameer, Phil McMinn, and William G.J. Halfond. 2018. Automated Repair of Internationalization Failures Using Style Similarity Clustering and Search-Based Techniques. In *Proceedings of the International Conference on Software Testing, Validation and Verification (ICST)*. 215–226. https://doi.org/10.1109/ICST.2018.00030
[33] Tuan A. Nguyen and Christoph Csallner. 2015. Reverse engineering mobile application user interfaces with REMAUI. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*. 248–259. https://doi.org/10.1109/ASE.2015.32
[34] Pavel Panchekha, Adam T. Geller, Michael D. Ernst, Zachary Tatlock, and Shoaib Kamil. 2018. Verifying That Web Pages Have Accessible Layout. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. 1–14. https://doi.org/10.1145/3192366.3192407
[35] Pavel Panchekha and Emina Torlak. 2016. Automated reasoning for web page layout. In *Proceedings of the ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, (OOPSLA)*. 181–194. https://doi.org/10.1145/2983990.2984010
[36] Rahul Pandita, Xusheng Xiao, Hao Zhong, Tao Xie, Stephen Oney, and Amit Paradkar. 2012. Inferring Method Specifications from Natural Language API Descriptions. In *Proceedings of the International Conference on Software Engineering (ICSE)*. 815–825. https://doi.org/10.1109/ICSE.2012.6227137

[37] Eric Sven Ristad and Peter N Yianilos. 1998. Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 20, 5 (1998), 522–532. https://doi.org/10.1109/34.682181

[38] Edward Rosten, Reid B. Porter, and Tom Drummond. 2010. Faster and Better: A Machine Learning Approach to Corner Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 32, 1 (2010), 105–119. https://doi.org/10.1109/TPAMI.2008.275

[39] Fei Shao and Xusheng Xiao. 2021. WebEvo Project Website. https://github.com/webevoexp/webevo.

[40] John Slankas, Xusheng Xiao, Laurie A. Williams, and Tao Xie. 2014. Relation extraction for inferring access control rules from natural language artifacts. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*. 366–375. https://doi.org/10.1145/2664243.2664280

[41] Andrea Stocco, Rahulkrishna Yandrapally, and Ali Mesbah. 2018. Visual Web Test Repair. In *Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018)*. 503–514. https://doi.org/10.1145/3236024.3236063

[42] OpenCV team. 2017. OpenCV. http://opencv.org/.

[43] AIMDek Technologies. 2019. Robotic Process Automation. https://medium.com/@AIMDekTech/evolution-of-robotic-process-automation-the-path-to-cognitive-rpa-c3bd52c8b865.

[44] UiPath. 2020. UiPath: Robotic Process Automation. https://www.uipath.com/rpa/robotic-process-automation.

[45] w3c. 2017. XML Path Language (XPath). https://www.w3.org/TR/2017/REC-xpath-31-20170321/.

[46] w3c. 2020. Cascading Style Sheets. https://www.w3.org/Style/CSS/.

[47] Shengqu Xi, Shao Yang, Xusheng Xiao, Yuan Yao, Yayuan Xiong, Fengyuan Xu, Haoyu Wang, Peng Gao, Zhuotao Liu, Feng Xu, and Jian Lu. 2019. DeepIntent: Deep Icon-Behavior Learning for Detecting Intention-Behavior Discrepancy in Mobile Apps. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2421–2436. https://doi.org/10.1145/3319535.3363193

[48] Xusheng Xiao, Amit Paradkar, Suresh Thummalapenta, and Tao Xie. 2012. Automated extraction of security policies from natural-language software documents. In *Proceedings of the International Symposium on the Foundations of Software Engineering (FSE)*. 12. https://doi.org/10.1145/2393596.2393608

[49] Xusheng Xiao, Xiaoyin Wang, Zhihao Cao, Hanlin Wang, and Peng Gao. 2019. IconIntent: Automatic Identification of Sensitive UI Widgets based on Icon Classification for Android Apps. In *Proceedings of the International Conference on Software Engineering (ICSE)*. 257–268. https://doi.org/10.1109/ICSE.2019.00041

[50] Tom Yeh, Tsung-Hsiang Chang, and Robert C. Miller. 2009. Sikuli: using GUI screenshots for search and automation. In *Proceedings of the Annual ACM Symposium on User Interface Software and Technology (UIST)*. 183–192. https://doi.org/10.1145/1622176.1622213