TOWARDS CROSS-BROWSER INCOMPATIBILITIES DETECTION: A SYSTEMATIC LITERATURE REVIEW

Willian Massami Watanabe[1], Fagner Christian Paes[2] and Daiany Silva[3]

[1,2,3]Universidade Tecnologica Federal Do Parana, Brazil

ABSTRACT

The term Cross Browser Incompatibilities (XBI) stands for compatibility issues which can be observed while rendering a web application in different browsers, such as: Microsoft Edge, Mozilla Firefox, Google Chrome, among others. In order to detect Cross-Browser Incompatibilities - XBIs, Web developers rely on manual inspection of every web page of their applications, rendering them in various configuration environments (considering multiple OS platforms and browser versions). These manual inspections are error-prone and increase the cost of the Web engineering process. This paper has the goal of identify- ing techniques for automatic XBI detection, conducting a Systematic Literature Review (SLR) with focus on studies which report XBI automatic detection strategies. The selected studies were compared accord- ing to the different technological solutions they implemented and were used to elaborate a XBI detection framework which can organize and classify the state-of-the-art techniques and may be used to guide future research activities about this topic.

1. INTRODUCTION

Web applications are built on top of a client-server architecture, in which the application is partially executed in the server-side through a variety of programming languages, while other components are executed in the client-side and can be rendered in different web browsers, such as: Internet Explorer, Microsoft Edge, Mozilla Firefox, Opera, Google Chrome, and others [1]. Thus, in order to execute the client-side components, users can choose from a broad variety of distinct browser implementations. While this flexibility work towards the portability of web applications, it also increases the cost and effort spent in the life cycle of the software development.The consistent rendering of a web application in different browsers is an essential attribute for high profile web-sites [2]. Every element of a web application should be correctly rendered and present the same behavior, regardless of the user's web browser implementation, version, or OS [3]. In this con-text, recent studies [1, 4] have referred to the incompatibilities observed in web applications when rendered by different browsers as *XBI* (Cross-Browser Issues/Incompatibilities).

To detect XBIs, developers must load and render web applications in multiple browsers, then man-ually inspect their behavior. Currently, there are commercial tools which can be used to alleviate the costs of this manual ad-hoc inspection activity, such as Microsoft Expression Web[1] and Adobe Lab[2]. These tools automatically generate screenshots of a web application as rendered by multiple browsers. However, even though they make the process of identifying XBIs easier, developers still have to rely on manual inspection of the screenshots.

According to Choudhary et al., XBIs can be classified in two groups [1]:

Layout Issues: caused by differences in how web browsers initially render a webpage; this type of XBI is immediately visible to the user, appearing as different positioning, size, visibility or general appearance of elements of a webpage; and

DOI: 10.5121/ijsea.2019.10602

Functionality Issues: associated to the functionality/behavior of a web application and the way it is executed in different browsers.

The state-of-the-art research in this topic reports on various experimental tools to automatically detect XBIs, such as WebDiff [1], CrossCheck [4], X-Pert [2], WebMate [3] and Browserbite [5]. These tools employ different techniques to implement a XBI detection solution, and were separately developed with the objective of reducing false positives (XBIs not actually observed in the web application) and increasing the number of detectable incompatibilities (associated to Layout and Funcionality Issues).

This paper presents a *SLR* (Systematic Literature Review) [6] to identify different approaches to XBI automatic detection. The results show the evolution of these approaches and classify the studies' different techniques, comparing them and synthesizing contributions in this topic to hopefully guide future efforts in automatic XBI detection. The contributions associated to this study are: (1) the elaboration of a XBI detection framework for organizing and classifying the technical solutions used in the studies; (2) a comparison between the different technical solutions employed to alleviate XBIs detection costs throughout the Software Engineering process; and (3) a SLR that can be referred to again in the future so as to evaluate the evolution of research in the area of automatic XBI detection.

This paper is structured as follows: Section 2 presents the process of Systematic Literature Review and its protocol; Section 3 presents the results of the SLR, with the classification of the XBI automatic detection approaches and description of the proposed framework; Section 4 presents related work and a discussion on this paper contributions; finally, Section 5 presents final remarks and suggests guidelines for future works.

2. SYSTEMATIC LITERATURE REVIEW

A SLR, also known as Systematic Review, is a secondary study which aims to identify, access, evaluate, and interpret all of the relevant studies in an attempt to answer certain Research Questions (RQs), shedding lights on topics or phenomena of interest [6]. SLRs are focused on gathering and synthesizing evidence regarding pre established research topic; their results are supposed to uncover the most current, state-of-the-art research on said topic.

According to Kitchenham et al., the SLR process is three-fold [6]:

- Review planning: clarification of key factors. Research questions and a detailed review protocol are produced. The review protocol should establish the goal, search method, search strings, and inclusion/exclusion criteria;

- Review conduction: search at selected sources, selection of studies based on the inclusion/exclusion criteria and data extraction;

- Review documentation: interpretation and reporting of results, in order to present them to interested parties. This can involve descriptive statistics, demographic information, visual information, and several discussions.

This SLR study identified automated approaches to XBI detection. We hereby present a compilation of our SLR results.

2.1. Review Protocol

This section presents the Review protocol which was used to conduct the SLR. The protocol attributes are presented next:

Objective: This study's goal was to identify approaches for the automated detection of XBIs, highlighting their strengths and weakness. Additionally, our results support a wide analysis of the evolution of automated XBI detection approaches, providing guidelines for future research on the topic.

Research Question (RQ): *What techniques have been employed by XBI detection approaches and how do they compare to one another?*

Search Method: The research method adopted by this study consisted of using a generic search string for searching different indexed research databases available online. This generic search string was defined as follows:

- *Keywords: ((“cross-browser" OR “cross browser") AND (compatibility OR incompatibility OR testing OR test))*

We performed searches on three different scientific databases: ACM Digital Library[3], IEEE Xplore[4], and Springerlink[5]. The search string was calibrated to each library according to the particular rules of its search engine.

We also emplyed a *forward snowball* technique, which consists of searching for articles that cite the review's previously selected articles [7]. This was done through Google Scholar[6] and IEEEXplorer.

Inclusion/Exclusion criteria: Aiming at answering our RQs and based on the scope of our research, we formulated the following inclusion/exclusion criteria:

- Inclusion criteria 1 (IC1): studies addressing the problem of automated detection of XBIs which employed *Generation 3 - Crawl and capture approaches* [2] were included in the SLR.
- Exclusion criteria 1 (EC1): studies addressing approaches for preventing XBIs during the coding phase of the development process were excluded;

- Exclusion criteria 2 (EC2): repeated entry (seen in more than one database) were excluded; and
- Exclusion criteria 3 (EC3): studies which did not mention XBI detection were excluded.

2.2. Conducting The Review

The review process is shown in Figure 1.

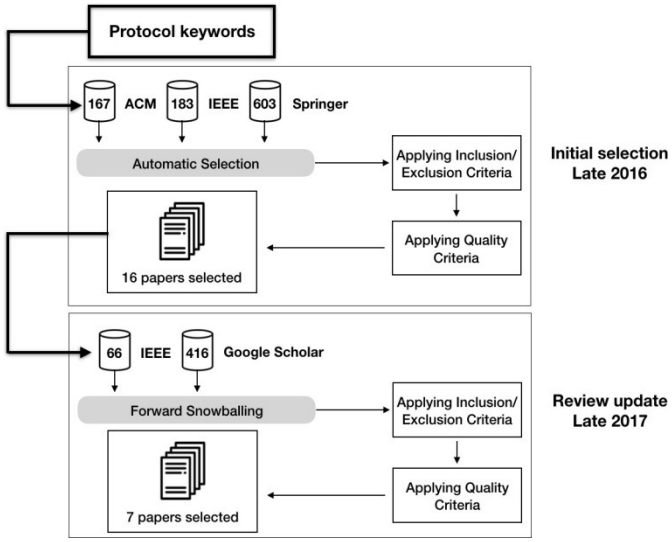

Figure 1. SLR process with the initial selection and forward snowballing results.

Our generic search string was adapted to execute properly in the search engine of the selected databases. We have conducted a two-step search: our initial search was executed on September 2015, and updated on August 2016. This returned a total of 953 studies (167 from ACM, 183 from IEEE Xplore, and 603 from Springer). After applying the inclusion and exclusion criteria to the studies, 16 studies were seleted.

After initial selection, the forward snowballing technique was performed. The search for studies which referenced the previously selected 16 studies was conducted in August 2017, and returned 482 studies (416 from Google Scholar and 66 from IEEE Xplore). These 482 studies were subjected to the same inclusion, exclusion criteria; this resulted in the selection of 7 more studies, in a total of 23.

The first selected studies in the SLR were published in 2010, while the year which presents the highest number of publications was 2014. Only 4 studies from 2017, were selected. However, it is unlikely that all publications from 2017 were included in the search, considering the last search update was performed in August 2017.

The next section presents the results of the SLR, with a detailed analysis of the identified automated approaches.

3. RESULTS

The 23 selected articles in the SLR reported 8 different tools for detecting XBIs. There were multiple articles which reported different experiments and evaluations conducted using the same tool, but there were also studies that, although describing technological solutions for the identification of XBIs, did not named their approach.

The technological approaches identified in this review were: **Structural DOM analysis**, **Screenshot comparison**, **Isomorphism in graphs**, **Machine learning**, **Relative layout**, **Adaptive Random Tests,** and **Record-n-play**. Figure 2 illustrates the number of articles which used each technological approach.

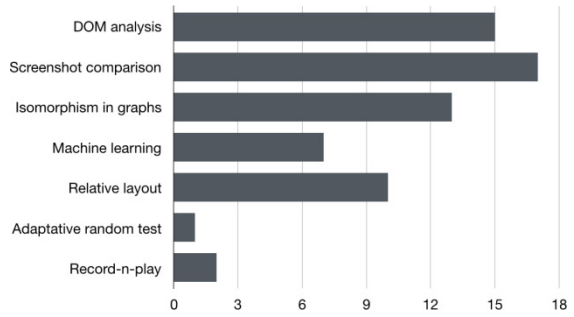

Figure 2. Number of articles which implement each technological approach to detect XBIs.

3.1. XBI Detection Framework

Based on the XBI detection approaches identified in this paper, we propose a general frame- work for classifying and organizing XBI detection approaches in the state-of-the-art and future researches in this area. Our proposed framework is illustrated in Figure 3. Design decisions made for modeling the framework were based on the analysis of XBI detection approaches proposed in the selected papers. The components of the framework were categorized according to two aspects: the type of the technical solution used and the XBI detection task.

In the framework we categorize XBI detection approaches in two main groups: DOM-based ap-

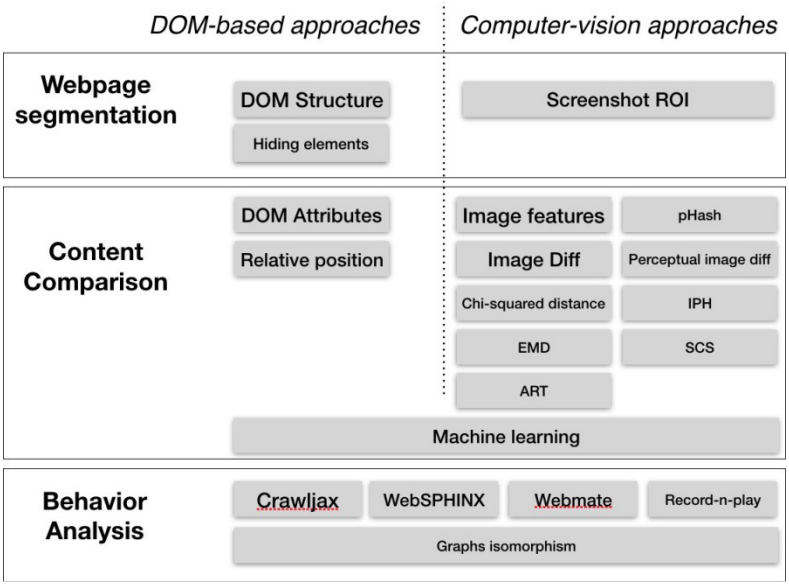

Figure 3. The XBI detection framework composed by webpage segmentation, content comparison and behavior analysis techniques.

proaches and Computer-vision approaches. The DOM-based group of approaches involves detection mechanisms which use the DOM structure of a web application for identifying XBIs. These approaches are mainly based on the Selenium API for navigating the DOM tree, collecting DOM attributes such as position and size of elements, and rendering the web application in multiple browsers. Computer-vision approaches are based mainly on image processing techniques applied to the complete screenshot or to screenshots of an specific region or element of the web application.

Many of the reviewed studies relied on DOM-based approaches (such as [8] which reported the use DOM elements' attributes for detecting XBIs) or Computer-vision approaches (such as [5, 9–11] which developed image processing algorithms for detecting XBIs). However, there were also hybrid approaches, which combined DOM-based and Computer-vision approaches (such as [4, 12] which used the DOM attributes alongside screenshot comparison for detecting XBIs).

In the studies, we observed the use of different approaches for detecting Layout and Functionality incompatibilities. In the framework, we propose decomposing the XBI detection process in three separate tasks: Webpage segmentation, which breaks the web application in smaller units of content before analyzing them for XBI detection; Content comparison, which employs techniques for comparing units of content as rendered by different browsers, detecting Layout incompatibilities; and Behavior analysis, which employs techniques for searching for Functional XBIs.

The Webpage segmentation task involved two main approaches, as observed in the reviewd studies. The first one was based on segmenting web applications according to their DOM structure, having as units of content each constituent DOM element. The second approach was based on segmentation using Regions-Of-Interest (ROI) identified from the complete screenshot of the web application. It is worth noting that the use of a Webpage segmentation technique also implies the use of a segment matching technique for comparing a segment as rendered by different browsers. Both approaches present specific segment matching strategies, based on X-Path, DOM attributes, position, size, and other characteristics extracted from the DOM structure or the screenshot image. There were also studies which did not report using any techniques associated to the Webpage segmentation task; in these cases, the XBI detection mechanism was based on the complete screenshot of the web application, such as [10], or the segmentation task was not the focus of the study [13, 14]. DOM-based webpage segmentation also allows the use of the crawling approach proposed in [15, 16]. In this approach, crawling process starts from leaf nodes of the DOM structure and hiding elements after collecting their data. Paes (2017) suggests this approach would increase the precision of the XBI detection approach. Furthermore, DOM-based Webpage segmentation is also present in other works associated to the automatic correction of XBIs, using search-based approaches [17].

The Content comparison task has the goal of identifying mainly Layout XBIs by comparing two or more corresponding elements/regions as rendered by different browsers. Our framework presents the techniques and characteristics which could be used for comparing these elements/regions:

- **DOM-based**
  - DOM Attributes, position, size, text content, extracted from the DOM structure [1, 18].
  - Relative position of parent and sibling nodes; building alignment graphs and compar- ing the same graph rendered in different browsers [2, 12, 15, 16, 19, 20].
- **Computer-vision**
  - Image features extracted from screenshots of the segments which compose the web- page [5, 9, 11].
  - Image Diff metric based on counting the number of dissimilar pixels between segments [15, 16].
  - EMD and $\chi 2$ distance of the histograms [4].

- pHash and Perceptual Image Diff algorithms [10].
- ART for optimizing the image comparison performance [13].
- IPH and SCS for using specific segment screenshot features for identifying incompatibilities [14].

In regards to the Content comparison task, several studies have reported using many of the previously defined techniques concurrently, in order to improve the effectiveness of their approaches. In these studies, Supervised Learning Classification algorithms were employed for merging the results of the different techniques into a single output, and classify the segments as containing XBIs or not.

The last part of the proposed framework is the Behavior analysis task. Components for detect- ing Functional XBIs were mapped. All approaches identified by the review to execute this task propose building a navigation/interaction model for the webpage, similar to a finite state machine graph [3, 4, 8, 19–23], then comparing these models as rendered by different browsers, using a graph isomorphism algorithm strategy. According to our framework, four technical approaches were used for generating these models: Crawljax [4, 8], Webmate [3, 22, 23], Record'n'play [19, 20], and WebSPHINX [24].

The proposed framework presents all technological solutions employed in the state-of-the-art in XBI detection approaches. It could be used to organize XBI detection solutions, experiment with multiple combinations of techniques and guide future research in the area of XBI detection.

3.2. Technical Solutions For XBI Detection

The technical solutions hereby presented are chronologically ordered according to when they were first proposed in the literature. A low-level individual analysis of these technical solutions provide insights on specific configuration scenarios and types of XBIs which were identified by each approach, highlighting the advantages of using each technique separately, in an attempt to answer **RQ**.

3.2.1. Structural Dom Analysis

The DOM (*Document Object Model*) defines a logical structure for HTML and XML documents, in order to allow programs and scripts to dynamically access and update the content[7]. The DOM represents HTML and XML documents as tree structures, and browsers frequently provide an API(Application Programming Interface) to be used by web applications. Many studies selected in the SLR process used the DOM structure of a webpage to detect XBIs. These studies compared attributes and elements of DOM representations as rendered by different browsers.

Initially, multiple DOM representations of a single webpage are captured. Then, these DOM representations are compared, in order to identify attributes and elements which do not present the same value [1–4, 8, 12, 18, 21–23, 25, 26]. In these approaches, if an element presents incompatible attribute values in different browsers, it can be marked as a XBI and later reported to developers. Most studies used a selected group of attributes verifications in this step.

The first article to use this approach was [1], in the WebDiff tool. It was further developed in other articles, such as [2, 4, 8]. While the studies evolved through time, it was observed that

simply comparing the attributes of the DOM structure as rendered by different browsers leads to a high number of false positives, representing XBIs detected by the approaches which were not observable in the web application decreasing the precision of the approach. Thus, studies using this technique advanced, employing other strategies such as attribute value normalization [4], machine learning [4, 15, 16], among other techniques which are described in the next sections of this paper.

Still in respect to structural DOM analysis, the studies [1, 2, 4] detalied solutions for matching elements from different DOM representations. DOM representations rendered by multiple browsers might present structural differences which might make it difficult to compare attributes and elements which can be miss placed depending on the browser in which it was rendered. Therefore, these studies presented a technique based on using attributes such as *tagName*, *id*, *xpath* and a hash mapping of the inner content (*innerHTML*), in order to match similar elements from different DOM representations.

Paes (2017) also proposed a specific DOM crawling technique, in which the DOM is crawled starting by the leaf nodes and then moving upward in the DOM tree. After collecting features for analysis of XBIs presence in a target DOM element (such as position and screenshot), that DOM element is hidden in the web application, so that analysis of other DOM elements is not impacted by the visual characteristics of the previously collected ones [15, 16].

3.2.2. Screenshot Comparison

DOM structure information can be used to assist the process of identifying XBIs. However, the DOM does not present information about each element's exact appearance on the screen [1]. Thus, in regards to the identification of Layout XBIs, many studies reported the use of screenshot comparison techniques [1, 2, 4, 5, 9, 10, 12, 13, 18, 25].

The primary method for comparing images is direct matching histograms. However, using a rigid comparison of histograms with no consideration for global features (such as symmetry, kurtosis and number of peaks) might raise many false positives [9]. In this context, image comparison techniques were used in the studies: Earth Mover's Distance (EMD) [1, 18, 25], $\chi 2$ Distance [2, 4, 12], Image Segmentation with correlation-based image comparison [5, 9], pHash and Perceptual Image Diff [10], Iterative Perceptual Hash, and Structure-Color-Saliency [14].

WebDiff employed EMD to individually compare screenshots taken in different browsers for each element of the DOM structure of a webpage [1, 18, 25]. In the WebDiff tool, there was no fixed threashold to determine whether two element screenshots were different or not [1]. Instead, the threshold was set relative to the size and the color density of each screenshot.

The $\chi 2$ distance image comparison technique is adapted from the statistical $\chi 2$ hypothesis test, which is used to compare the dispersion between two nominal variables, evaluating the correlation between qualitative variables. The studies [2, 4, 12] used the $\chi 2$ distance method to compare screenshots of elements of the webpage. In [2, 12], the comparison was carried out between all elements of a webpage. Then, in a later study [4], the $\chi 2$ distance method was used to compare only the leaf nodes of the DOM tree structure. This strategy was used to reduce XBI false positives. In [2, 12], the authors also reported that the $\chi 2$ is less expensive computationally and more precise than EMD.

The studies [5, 9], which presented the Browserbite tool, reported segmenting screenshots of web pages as rendered by different browsers into small rectangles, each identified as a Region Of Interest (ROI) based on the image's borders and color changes. Screenshots of the ROI as captured

by different browsers were then compared. These studies employed a correlation-based image comparison technique [5].

Sanchez and Aquino Jr. used two approaches for screenshot comparison: pHash[8] and Perceptual Image Diff[9] [10]. The pHash algorithm is an implementation of the Perceptual Hash concept, defined as a fingerprint hash of a multimedia file derived from various features of its content[10]. Perceptual Image Diff, on the other hand, is an algorithm to identify every dissimilar pixel between two images. Firstly, pHash was applied to screenshots captured from different browsers, in order to generate a hash code for each image. Subsequently, the hash code of each image was compared using the Hamming distance algorithm. The result was used alongside the Perceptual Image Diff algorithm to detect XBIs.

In [14], two screenshot comparison algorithms were proposed: IPH - *Iterative Perceptual Hash* and SCS - *Structure-color-saliency*. IPH uses global and local content discrepancies to avoid under or overestimation of changes between screenshots. SCS considers changes in color, structure, saliency and location sensitivity to compose a unique index. These approaches were used to analyze differences between screenshots of parts of the webpage. In the experiment conducted by the authors, IPH outperformed other previously used techniques, such as EMD (CrossCheck [4]) and $\chi 2$ (X-Pert [12]).

3.2.3. Graphs Isomorphism

The studies [2–4, 8, 12, 21–23, 26] reported the use of different approaches which focused on the identification of **Functionality** XBIs. These approaches consisted of simulating user interac- tion scenarios, mainly click events on elements of the webpage, to trigger changes in the DOM representation of the web application in different browsers. The These studies mapped the DOM representation and click event simulations into a graph, and then used a Graph Isomorphism algorithm to identify functionalities which were not available depending on the user's browser.

The first study to use this approach was in CrossT [8]. This study extended an Ajax application crawler tool, named Crawljax [27]. CrossT employed the Crawljax for representing the Navigation model of the application. Then, the Navigation models rendered by different browsers were compared and differences in these graphs were reported as **Functional** XBIs, i.e., click functionalities which were observed in one browser and missed in another [8].

After CrossT, other XBI detection approaches implemented similar strategies, such as [2–4, 12, 19–23, 26]. Certain approaches reported using an equivalent strategy for generating the models [2, 4, 12], other reported doing so through events besides click [3, 21–23] and Record'n'play strategies for also mapping non-deterministic events [19, 20]. In these approaches, the studies evolved with the goal of increasing the number of DOM structure states analysed and covered by their XBI detection technique.

3.2.4. Machine Learning

Machine learning is an area of Artificial Intelligence which has the objective of developing computational techniques capable of automatically acquiring knowledge on how to perform a specific task without being explicitly programmed to do so. All XBI detection studies which reported the use of machine learning used supervised learning techniques. In supervised learning, concepts are induced into the system from pre-classified examples. Studies employed different character- istics of web applications as features in the learning phase and also different machine learning algorithms.

The first studies which used machine learning techniques for XBI detection were [2, 4, 12], in CrossCheck and X-Pert. These studies reported the use of the decision-tree machine learning technique. A decision-tree is a simple representation of a classifier used by many machine learning systems, such as C4.5 [28]. These studies used a decision tree to classify every element of the DOM structure as rendered by different browsers and determine if it was a XBI or not. The decision tree, in these articles, was modelled with the following features: **Size difference ratio**, differences in the size of elements as rendered by different browsers; **Displacement**, the Euclidean distance between the position of elements as rendered by different browsers; **Area**, the smaller area of elements as rendered by different browsers; **Leaf Node Text Differences**, a boolean value identifying whether there are differences between the texts of elements as rendered by different browsers; and **Image distance**, the $\chi 2$ distance screenshot comparison of elements as rendered by different browsers. A simplified classification model of DOM elements was reported in [15, 16], making use of the following features: Displacement, Relative position of the element in regards to its parent position, Size differences, and image distance of the screenshots.

After [2, 4, 12], in the Browserbite tool, the authors reported the use of decision trees and neural networks to detect XBIs in web applications [5, 9]. Neural networks are mathematical models inspired by organic neural structures, which acquire knowledge through experience [29]. The validation of Browserbite consisted of an experiment to compare how both machine learning approaches performed when analyzing a group of websites. Moreover, Browserbite uses the concept of regions of interest to identify XBIs [5, 9], differently from other approaches, which employed DOM elements comparison. Browserbite's neural network approach was applied to each region of interest as rendered by two browsers using the following features: **10 histogram bins** (h0, h1, ..., h9) representing the pixel intensity distribution of the region of interest in the baseline browser; **Correlation-based image comparison between screenshots of region of interest** as rendered by two browsers; **Horizontal and vertical position** of the region of interest as rendered by two browsers; **Height/Width** of the region of interest as rendered by two browsers; **Configuration index** indicating the browsers which were used to render the regions of interest; and **Mismatch density metric**, calculated as a coefficient of the number of regions of interest which presented differences in the correlation-based image comparison approach and the total number of regions of interest in the screenshot of the webpage.

Decision trees and neural network machine learning techniques were used to reduce the number of false positive. When evaluating CrossCheck and X-Pert, the authors reported that the use of decision trees improved the results of their XBI detection approach. In the evaluation of Browserbite, neural networks performed better in comparison to decision trees [5, 9]. However, it is difficult to establish a comparison between [2, 4, 12] and [5, 9], since these studies used a different set of features and training data.

3.2.5. Relative Layout Comparison

The WebMate tool compared elements considering their relative positions [3, 22, 23]. HTML elements present a hierarchical structure for rendering the layout of a webpage; hence, if a parent element is misaligned, its child elements will also present the same misaligned absolute positioning. Thus, XBI detection approaches using absolute positioning to identify XBIs, might report that child elements represent XBIs, when only the parent element was actually misaligned.

X-Pert also explored the relative layout position evaluation strategy [2, 12]. The tool modeled the positioning of elements in a webpage as a graph, titled *Alignment Graph*. The Alignment Graph represents the relationships between parent, child and siblings nodes, represented as nodes in the graph, and their relative position in regards to one another, represented as transitions between each element in the graph. Two alignment graphs extracted from two different browsers can then be

checked for equivalence, and any difference observed between the graphs is reported as a XBI.

3.2.6. Adaptive Random Testing (ART)

Adaptive Random Testing - ART is an extension of the *Random Testing* technique. ART analyses the behavior of test results observed in software and generate test input which are more likely to produce failures in the software [30]. Generally, similar test cases tend to identify similar failures, hence ART is a technique to identify test cases which are more likely to improve the coverage of a software input domain.

Selay et al. [13] used ART to compare images for equivalence, employing a screenshot comparison approach. The ART approach used in [13] implemented the *Fixed Size Candidate Set (FSCS)* algorithm to randomly chose test cases. This algorithm makes use of a test case similarity metric to guarantee that it generates different test cases to explore different parts of the software's input domain [30]. Selay et al. ran an experiment with the goal of improving the performance in terms of the time required to identify incompatibilities while comparing two distinct screenshots [13].

3.2.7. Record'n'play

In order to detect Functional XBIs, the approaches built a navigation graphs for each browser and carried out further equivalence analysis of these graphs. Initially these graphs were built from data generated by AJAX crawling tools, such as Crawljax and WebMate, which simulate user interactionscenarios. However, these approaches do not deal with non-deterministic events in the web application, or dynamic content changes generated by the server.

In [19, 20], the use of a proxy-based approach is proposed for recording web application's dispatched events and replaying the same exact events when rendered in different browsers, through the X-Check tool. Besides increasing the precision of the XBI detection approaches, this allowed the analysis of specific user interaction scenarios, considering events not priorly programed in other crawlers, possibly enhancing the coverage of web applications. In [19, 20], X-Check is compared to the X-Pert tool, in which the X-Check showed enhanced effectiveness and higher number of XBI detection than X-Pert.

4. RELATED WORK AND DISCUSSION

There are other secondary studies in the area of Web applications testing [31–33]. Garousi et al. (2013) conducted a Systematic Mapping study [31] for identifying and classifying research effort in this area. Their study reported 79 papers from 2000 to 2011 and their classification considered aspects such as Testing activity, Testing location, Generated Artifacts, Used Techniques, Static/dynamic webpages testing, Synchronous/Asynchronous, Client-tier technologies, Server-tier technologies, among other data.

Li et al. (2014) presented a survey to report the evolution of techniques throughout 20 years of Web application testing [32]. Their study classified articles according to Motivation (Interoperability, Security, Dynamics, among others), Graph and model-based techniques, Mutation testing, Search-based Software Engineering testing, Scanning and crawling techniques, Random testing, Fuzz testing, Concolic testing and User-based testing.

Dogan et al. (2014) conducted a SLR on Web application testing [33]. The authors selected 95 papers published from 2000 to 2013. Their SLR was conducted with the goal of listing tools, identifying test/fault models used in the studies, map the empirical evaluation methods and their relevance for the industry.

This paper presented a SLR on strategies for automatically detecting XBIs in web applications. In the SLR, 23 studies were selected and 7 XBI detection strategies were identified in these stud- ies: Structural DOM analysis, Screenshot comparison, Graphs isomorphism, Machine learning, Relative layout comparison, Adaptive random testing and Record'n'play. We described these strategies, compared how each study implemented them and synthesized their results.

We also proposed a framework which separated XBI detection strategies into three main steps: Webpage segmentation, Content comparison and Behavior analysis. The 23 selected studies are situated in distinct parts of this framework, which could be used to organize these studies and future research in the area. The framework classifies XBI detection approaches as belonging to either of two groups: DOM-based or Computer-vision. The framework presents the components of each group and how these components could be used together to form a XBI detection approach. These results could also be used to guide future research in the area, and for comparing and implementing new approaches.

XBI detection strategies evolved through time, targeting both Layout and Functionality XBIs. Multiple approaches have been reported to improve the precision of XBI detection and the range of XBIs identified. Web applications consist of multiple web pages and complex built-in functionalities. The studies aimed to improve precision in XBIs identification and help decrease the software development costs. Most validation procedures conducted in the studies consisted of applying their XBI detection strategy to a group of websites, then comparing the effectiveness of their approach with the results obtained through manual inspection or with other approaches. Even though each experiment was conducted while measuring the effectiveness of the proposed XBI detection strategies, their results are not comparable, since they were analyzed with different websites. The number of articles in relation to each browser used in the experiments were: Mozilla Firefox (10 articles), Google Chrome (7 articles), Microsoft Internet Explorer (10 articles), and Apple Safari (4 articles). It is worth noting that in [11], the experiment was conducted with 15 distinct configurations of these browsers, by changing the OS and browser versions.

Even though browser engines have also evolved, research in this area still poses several challenges, such as: detecting cross-browser incompatibilities between different platforms (Cross-platform incompatibilities) [16, 34], given the increased popularity and variety of mobile devices and alternative operational systems; responsive design failure detection [35] for detecting inconsistencies in web applications when rendered in different viewport width scenarios; and automatic correction of Cross-browser incompatibilities using Search-Based Software Engineering [17].

The SLR process was conducted in order to identify all automatic XBI detection approaches which were used in the state-of-the-art research. However, there might be studies which report the use of XBI detection techniques but were not included. In order to aleviate this concern, the SLR process was conducted by two researchers; all studies and the applicability of inclusion/exclusion/quality criteria were discussed and reviewed by them. Futhermore, our SLR process used database searches in ACM, IEEEXplore and SpringerLink, and forward snowballing in IEEEXplorer and Google Scholar to search for articles related to XBI detection techniques. The results of this SLR could be enhanced if other databases were included and backward snowballing was also employed. However, the inclusion of other studies in the SLR would mean the addition of other approaches related to the already described ones in this SLR or, at most, the addition of a new approach category, not invalidating any of the previously reported results.

5. FINAL REMARKS

Cross Browser Incompatibilities (XBIs) are frequent in web projects and represent a serious problem for web application developers. This article conducted a Systematic Literature Review with the goal of identifying different approaches which have been used to automatically detect XBIs. The results showed that many studies used Structural DOM Analysis and Screenshot Comparison techniques for the automatic detection of XBIs. It was observed that most studies evolved with the goal of reducing the number of false positives reported using Machine learning, Graphs isomorphism, Relative layout comparison, ART and Record'n'play strategies. The techniques were also organized into a XBI detection framework composed of Webpage segmentation, Content comparison and Behavior analysis. The classification scheme of different technological approaches, as well as the comparison between the XBI detection strategies and the proposed framework, can be used to guide the elaboration of new strategies and tools.

Future studies should discuss the implementation of automatic XBIs detection for different platforms (on desktop and mobile devices), running experiments to compare the effectiveness of different XBI detection approaches and investigate techniques able to assist the developer in the diagnosis and correction of incompatibilities.

ACKNOWLEDGMENTS

The authors would like to thank UTFPR, CAPES and CNPq for supporting this work.

REFERENCES

[1] S. Choudhary, H. Versee, and A. Orso, "Webdiff: Automated identification of cross-browser issues in web applications," in *Proc. of the 2010 IEEE International Conference on Software Maintenance (ICSM 2010)*, Sept 2010, pp. 1–10.

[2] S. Choudhary, M. Prasad, and A. Orso, "X-pert: Accurate identification of cross-browser issues in web applications," in *Proc. of the 35th International Conference on Software Engi- neering (ICSE 2013)*, May 2013, pp. 702–711.

[3] V. Dallmeier, M. Burger, T. Orth, and A. Zeller, "Webmate: Generating test cases for web 2.0," in *Software Quality. Increasing Value in Software and Systems Development*, D. Win- kler, S. Biffl, and J. Bergsmann, Eds., vol. 133. Berlin, Heidelberg: Springer Berlin Hei- delberg, 2013, pp. 55–69.

[4] S. Choudhary, M. Prasad, and A. Orso, "Crosscheck: Combining crawling and differenc- ing to better detect cross-browser incompatibilities in web applications," in *Proc. of the 5thInternational Conference on Software Testing, Verification and Validation (ICST 2012)*, April 2012, pp. 171–180.

[5] N. Semenenko, M. Dumas, and T. Saar, "Browserbite: Accurate cross-browser testing via machine learning over image features," in *Proc. of the 29th IEEE International Conference on Software Maintenance (ICSM 2013)*, Sept 2013, pp. 528–531.

[6] B. Kitchenham, "Procedures for performing systematic reviews," Keele University, Keele, UK, Tech. Rep., 2004.

[7] D. Badampudi, C. Wohlin, and K. Petersen, "Experiences from using snowballing and database

searches in systematic literature studies," in *Proc. of the 19th International Conference on Evaluation and Assessment in Software Engineering (EASE 2015)*. New York, NY, USA: ACM, 2015, pp. 17:1–17:10. [Online]. Available: http: //doi.acm.org/10.1145/2745802.2745818

[8] A. Mesbah and M. R. Prasad, "Automated cross-browser compatibility testing," in *Proceedings of the 33rd International Conference on Software Engineering (ICSE 2011)*. New York, NY, USA: ACM, 2011, pp. 561–570. [Online]. Available: http://doi.acm.org/10.1145/1985793.1985870

[9] T. Saar, M. Dumas, M. Kaljuve, and N. Semenenko, "Cross-browser testing in browserbite," in *Web Engineering*, S. Casteleyn, G. Rossi, and M. Winckler, Eds. Cham: Springer Inter- national Publishing, 2014, pp. 503–506.

[10] L. Sanchez and P. T. A. Jr., "Automatic deformations detection in internet interfaces: Addii," in *Proceedings of the 17th International Conference on Human-Computer Interaction. Part III: Users and Contexts (HCI International 2015)*, ser. Lecture Notes in Computer Science, vol. 9171. Springer International Publishing, 2015, pp. 43–53.

[11] T. Saar, M. Dumas, M. Kaljuve, and N. Semenenko, "Browserbite: Cross-browser testing via image processing," *Softw. Pract. Exper.*, vol. 46, no. 11, pp. 1459–1477, Nov. 2016. [Online]. Available: https://doi.org/10.1002/spe.2387

[12] S. R. Choudhary, M. R. Prasad, and A. Orso, "X-pert: A web application testing tool for cross-browser inconsistency detection," in *Proc. of the 2014 International Symposium on Software Testing and Analysis (ISSTA 2014)*. New York, NY, USA: ACM, 2014, pp. 417–420. [Online]. Available: http://doi.acm.org/10.1145/2610384.2628057

[13] E. Selay, Z. Q. Zhou, and J. Zou, "Adaptive random testing for image comparison in regres- sion web testing," in *Proc. of the 2014 International Conference on Digital Image Comput- ing: Techniques and Applications (DICTA 2014)*, Nov 2014, pp. 1–7.

[14] P. Lu, W. Fan, J. Sun, H. Tanaka, and S. Naoi, "Webpage cross-browser test from image level," in *2017 IEEE International Conference on Multimedia and Expo (ICME)*, July 2017, pp. 349–354.

[15] F. C. Paes, "Automatic detection of cross-browser incompatibilities using machine learning and screenshot similarity: Student research abstract," in *Proceedings of the Symposium on Applied Computing*, ser. SAC '17. New York, NY, USA: ACM, 2017, pp. 697–698. [Online]. Available: http://doi.acm.org/10.1145/3019612.3019924

[16] F. C. Paes and W. M. Watanabe, "Layout cross-browser incompatibility detection using machine learning and dom segmentation," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, ser. SAC '18. New York, NY, USA: ACM, 2018, pp. 2159–2166. [Online]. Available: http://doi.acm.org/10.1145/3167132.3167364

[17] S. Mahajan, A. Alameer, P. McMinn, and W. G. J. Halfond, "Xfix: An automated tool for the repair of layout cross browser issues," in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2017. New York, NY, USA: ACM, 2017, pp. 368–371. [Online]. Available: http://doi.acm.org/10.1145/3092703.3098223

[18] S. Choudhary, "Detecting cross-browser issues in web applications," in *Proc. of the 33rd International Conference on Software Engineering (ICSE 2011)*, May 2011, pp. 1146–1148.

[19] G. Wu, M. He, H. Tang, and J. Wei, "Detect cross-browser issues for javascript-based web

applications based on record/replay," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Oct 2016, pp. 78–87.

[20] M. He, G. Wu, H. Tang, W. Chen, J. Wei, H. Zhong, and T. Huang, "X-check: A novel cross-browser testing service based on record/replay," in *2016 IEEE International Conference on Web Services (ICWS)*, June 2016, pp. 123–130.

[21] X. Li and H. Zeng, "Modeling web application for cross-browser compatibility testing," in *Proc. of the 15th IEEE/ACIS International Conference on Software Engineering (ICSE 2014), Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD 2014)*, June 2014, pp. 1–5.

[22] V. Dallmeier, M. Burger, T. Orth, and A. Zeller, "Webmate: A tool for testing web 2.0 applications," in *Proc. of the Workshop on JavaScript Tools (JSTools 2012)*. New York, NY, USA: ACM, 2012, pp. 11–15. [Online]. Available: http: //doi.acm.org/10.1145/2307720.2307722

[23] V. Dallmeier, B. Pohl, M. Burger, M. Mirold, and A. Zeller, "Webmate: Web application test generation in the real world," in *Proc. of the Workshops of the IEEE 7th International Con- ference on Software Testing, Verification and Validation Workshops (ICSTW 2014)*, March 2014, pp. 413–418.

[24] C. P. Patidar, M. Sharma, and V. Sharda, "Detection of cross browser inconsistency by comparing extracted attributes," *International Journal of Scientific Research in Computer Science and Engineering*, vol. 5, no. 1, pp. 1–6, 2017. [Online]. Available: http://www.isroset.org/journal/IJSRCSE/full_paper_view.php?paper_id=307

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

[26] H. Shi and H. Zeng, "Cross-browser compatibility testing based on model comparison," in *Computer Application Technologies (CCATS), 2015 International Conference on*, Aug 2015, pp. 103–107.

[27] A. Mesbah, A. van Deursen, and S. Lenselink, "Crawling ajax-based web applications through dynamic analysis of user interface state changes," *ACM Transactions on the Web*, vol. 6, no. 1, pp. 3:1–3:30, Mar. 2012. [Online]. Available: http: //doi.acm.org/10.1145/2109205.2109208

[28] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[29] J. Heaton, *Introduction to Neural Networks for Java, 2Nd Edition*, 2nd ed. Heaton Research, Inc., 2008.

[30] T. Y. Chen, F.-C. Kuo, R. G. 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, Jan. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2009.02.022

[31] V. Garousi, A. Mesbah, A. Betin-Can, and S. Mirshokraie, "A systematic mapping study of web application testing," *Information and Software Technology*, vol. 55, no. 8, pp. 1374 – 1396, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/ S0950584913000396

[32] Y.-F. Li, P. K. Das, and D. L. Dowe, "Two decades of web application testing-a survey of recent

advances," *Inf. Syst.*, vol. 43, no. C, pp. 20–54, Jul. 2014. [Online]. Available: http://dx.doi.org/10.1016/j.is.2014.02.001

[33] S. Doğan, A. Betin-Can, and V. Garousi, "Web application testing: A systematic literature review," *J. Syst. Softw.*, vol. 91, pp. 174–201, May 2014. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2014.01.010

[34] W. M. Watanabe, G. L. Amêndola, and F. C. Paes, "Layout cross-platform and cross-browser incompatibilities detection using classification of dom elements," *ACM Trans. Web*, vol. 13, no. 2, pp. 12:1–12:27, Mar. 2019. [Online]. Available: http://doi.acm.org/10.1145/3316808

[35] T. A. Walsh, G. M. Kapfhammer, and P. McMinn, "Redecheck: An automatic layout failure checking tool for responsively designed web pages," in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2017. New York, NY, USA: ACM, 2017, pp. 360–363. [Online]. Available: http://doi.acm.org/10.1145/3092703.3098221