Towards Cross-Browser Incompatibilities Detection: a Systematic Literature Review

Omitted due to blind revision 1, Omitted due to blind revision
Omitted due to blind revision 2, Omitted due to blind revision
Omitted due to blind revision 3, Omitted due to blind revision

Cross Browser Incompatibilities (XBI) stands for compatibility issues which can be observed while rendering the same web application in different browsers. Users can interact with the Web through distinct web browsers implementations, such as: Internet Explorer, Microsoft Edge, Mozilla Firefox, Opera, Google Chrome, among others. However, with the increasing number of browser implementations and the continually evolving characteristic of web technologies lead to differences in how browsers behave and render web applications. In order to overcome this issue during the software development process, web developers must detect and fix XBIs before deploying web applications, regardless of the effort and cost required to conduct these inspections. Many developers rely on manual inspection of every web page of their applications rendered in various configuration environments (considering multiple OS platforms, browser implementations and versions) to detect XBIs, independently of the effort and cost required to conduct these inspections. This paper presents a Systematic Literature Review (SLR) on XBI automatic detection strategies published as primary studies. The SLR process was conducted with the goal of identifying distinct techniques which have been used to identify XBI, present the evolution of the developed tools, and guide future research on the topic. In accordance to our findings, the strategies identified through the primary studies range from DOM (Document Object Model) Structure analysis, screenshot comparison, graph's isomorphism algorithms, machine learning, adaptive random testing, relative layout comparison, and static analysis.The SLR found 17 articles and 7 developed tools.

CCS Concepts: •**Information systems** → **Browsers;** •**Software and its engineering** → *Software notations and tools; Software creation and management;*

Additional Key Words and Phrases: Cross-Browser; XBI; XBI detection; Automatic Tools

ACM Reference Format:
Omitted, 2010. Towards Cross-Browser Incompatibilities Detection: a Systematic Literature Review. *ACM Trans. Embedd. Comput. Syst.* 9, 4, Article 39 (March 2010), 22 pages.
DOI: 0000001.0000001

1. INTRODUCTION

Web applications have been growing in complexity and interactivity, since the second generation of the Web, in the so called "Web 2.0". These applications are built based on a client-server architecture, in which the application is partially executed in the server-side through a variety of programming languages and other components are executed in the client side mainly through JavaScript inside web browser implementations, such as: Internet Explorer, Microsoft Edge, Mozilla Firefox, Opera, Google Chrome, among others [Choudhary et al. 2010b]. In order to execute the client-side components, currently, users can choose from a broad variety of distinct browser implementations. This flexibility which characterizes the portability of web applications, on the other hand, increases the cost and effort spent in the software development

Omitted due to blind revision

DOI: 0000001.0000001

life cycle, since one single web application must support multiple browsers. The task of ensuring that a web application is consistently rendered in all browsers or at least the most popular ones is an essential quality attribute expected for high profile websites [Choudhary et al. 2013]. Every element of a web application should be correctly rendered and presenting the same behavior, despite the web browser implementation, version, or OS which is used by users [Dallmeier et al. 2013]. In this context, in recent studies [Choudhary et al. 2010b; Choudhary et al. 2012], the incompatibilities observed in web applications when rendered in different browser implementation have been referenced as *XBI*, Cross-Browser Issues/Incompatibilities.

With the goal of detecting XBIs, developers must load and render web applications in multiple browsers, then manually inspect if the web applications is presented and behaves consistently, thus does not present XBIs. Currently, there are comercial tools which can be used to alleviate costs of this manual and ad-hoc inspection activity, such as Microsoft Expression Web[1] and Adobe Lab[2]. These tools automatically generate screenshots of a web application rendered in multiple browsers. However, even though these tools make the process of identifying XBIs easier, developers still rely on the manual inspection of the screenshots to detect XBIs. Henceforth, the detection of XBIs still poses significant effort and cost for the development process.

The state of the art in this topic reports various experimental tools to automatically detect XBIs, such as: WebDiff [Choudhary et al. 2010b], CrossCheck [Choudhary et al. 2012], X-Pert [Choudhary et al. 2013], WebMate [Dallmeier et al. 2013] and Browserbite [Semenenko et al. 2013]. This paper presents a *SLR - Systematic Literature Review* [Kitchenham 2004] which identify different approaches which were implemented to automatically detect XBIs. The results show the evolution of the approaches and classify the studies approaches, comparing them and synthesizing contributions in this topic to guide future efforts in XBI automatic detection. The contributions associated to this study are: (1) a taxonomy and classification of the approaches to automatically identify XBIs; (2) an overview of the supporting tools that can be employed in new approaches to alleviate XBIs issues; and (3) an SLR that can be loaded again in the future to evaluate the evolution of this research.

These tools propose the use of DOM Structure Analysis, Screenshot Comparison, Isomorphism Graph Algorithms, Machine Learning, Adaptive Random Testing, Static Analysis and Relative Layout Comparison to detect XBIs. These approaches evolved over the last years and they were separately implemented in articles with the objective of reducing false positives in the automatic detection approach (XBIs identified by the tools which were not observed in the web application). False positives in XBI detection tools represent issues which must be manually evaluated by web developers, thus impact negatively in the development process.

In comparison to other literature reviews conducted on Web applications testing approaches (such as: [Garousi et al. 2013; Doan et al. 2014]), this paper presents a Systematic Literature Review focused on Cross-Browser compatibility detection tools and the technical approaches which enable XBI automatic identification.

This paper is structured as follows: Section 2 introduces the XBI concept and describes the generations of XBI detection techniques; Section 3 presents the process of Systematic Literature review and the protocol; Section 4 presents the results of the SLR, with the classification of the XBI automatic detection approaches; Section 5 presents threats to validity; and Section 6 presents final remarks and future works.

[1]see: https://www.microsoft.com/expression/eng/
[2]see: http://labs.adobe.com/

2. XBI DETECTION

XBI detection consist of distinct technological solutions which were used to identify incompatibility in web applications when rendered in multiple browsers. A simple webpage is built based on three main client-side technologies: the HTML (HyperText Markup Language) which markup the structure of a text, the CSS (Cascading Style Sheets) which represents layout information of a webpage and JavaScript which handles the dynamic behavior and interactivity of a webpage.

XBI represents features in each of these technologies (HTML, CSS and JS) which are not supported consistently in web browsers and if a web application makes use of these features, then it will present an XBI. Figure 1 illustrates a webpage which is rendered differently in three distinct browsers: the Internet Explorer, Google Chrome, and Firefox. This incompatibility is a result of the fact that even though there are several browser specifications which should guide their development, there still remain gaps and requirements which can be implemented differently by development teams.

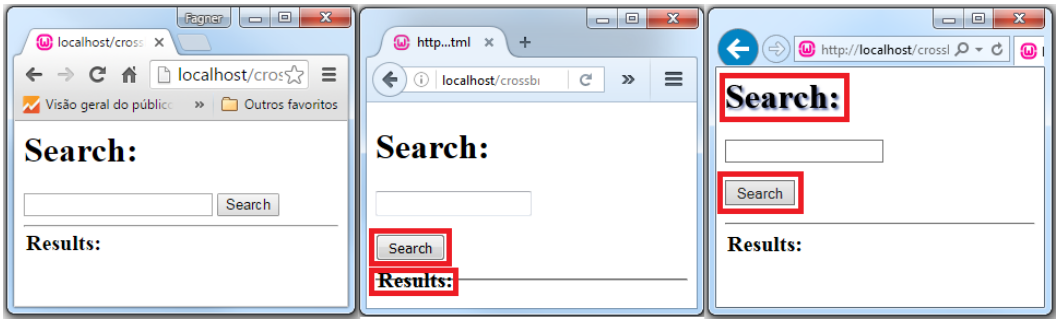


Fig. 1. A single webpage based on HTML/CSS/JS which is rendered differently in the Internet Explorer, Google Chrome, and Firefox.

According to Choudhary et al., XBIs can be classified in two groups [Choudhary et al. 2010b]:

Layout Issues:. are caused by differences in how web browsers initially render a webpage and are immediately visible to the user. These issues can be a result of different positioning, size, visibility or general appearance of elements of a webpage; and

Functionality Issues:. these issues are associated to the functionality/behavior of a web application and the way it is executed in different browsers. These issues might limit the way users interact with a web application depending on the browser the user is using.

Through the development process, developers can employ a variety of techniques to identify XBIs in web applications. Choudhary et al. classify XBI detection techniques in four generations [Choudhary et al. 2013]:

Generation 0 - Developer tool support. a reference table of HTML/CSS/JS features which are known to cause XBI in web applications, such as: QuirksMode[3] and Can I use[4]. While using Generation 0 XBI detection strategies, web developers can manually inspect the reference table to identify which features can be used and which

[3]http://www.quirksmode.org/
[4]http://caniuse.com/

should be avoided while coding their web application. The reference tables are manually built and maintained. Generation 0 XBI detection strategies, however can only identify XBI associated to the use of single features. Many XBIs reported in web applications can be associated to the use of multiple features combined in a single web page in multiple browsers.

Generation 1 - Test on a single browser. techniques which identify faulty presentation in a single browser. Eaton and Memon elaborated an approach to detect potentially problematic HTML tags on a web page based on empirical data from fielded systems [Eaton and Memon 2007]. Choudhary et al., however, report that the technique is human intensive and not well suited for modern web applications with a large number of dynamically generated content [Choudhary et al. 2013].

Generation 2 - Multi-platform behavior and test emulation. introduces the emulation of multi-platform environments for web applications. These tools render a web application in multiple browser engines, log how resources were loaded and take screenshots of the web application. Examples of tools of the Generation 2 of XBI detection strategies are the Microsoft Expression Web[5] and Adobe Lab[6]. Even though these tools ease the XBI detection process, they do not automatically identify XBIs. In these tools, web developers must manually evaluate resources loading logs and screenshot images to search for XBIs.

Generation 3 - Crawl and compare approaches. this generation of tools consists of techniques which automatically identify XBIs for web developers, with no need of human interventions on the process. These tools' XBI detection process consists of two steps [Choudhary et al. 2013]: the **behavior capture**, which consists of capturing characteristics of an web application when rendered in multiple browsers, such as DOM attributes and screenshots; and the **behavior comparison**, which consists of comparing the captured characterists in multiple browsers to identify XBIs.

The Systematic Literature Review conducted in this article focuses on *Generation 3 - Crawl and compare* XBI detection approaches.

3. SYSTEMATIC LITERATURE REVIEW

A SLR, also known as Systematic Review, consists of a secondary study whose goals are identify, access, evaluate, and interpret all of the relevant studies in accordance with some Research Questions (RQ), topic or phenomenon of interests [Kitchenham 2004]. SLRs are focused on gathering and synthesizing evidence in a pre-established research topic and their results are supposed to determine the state-of-the art for that topic. In the SLR context, "primary studies" are original studies whose results contribute to the purpose of a particular SLR [Budgen and Brereton 2006]. Due to this, SLRs are commonly referred as "secondary studies". An SLR must be conducted following some pre-defined rules that are stated in a research protocol. Thus, through the conduction of an SLR, researchers and practitioners are supposed to identify all of the relevant studies in a research topic following some organized and judicious sequence of steps. This leads the SLR process to configure an auditable-replicable method, being one of the most riskless strategies to start a novel research and allowing the assessment of the most useful scientific knowledge on the research topic. In accordance to [Kitchenham 2004], the central reasons for conducting SLRs are: *(i)* to obtain a birds-eye view over some technology, e.g. summarizing the empirical evidences on the pros and cons of using an agile-based specification method; *(ii)* to identify potential lacks of research

[5]see: https://www.microsoft.com/expression/eng/
[6]see: http://labs.adobe.com/

in contemporary topics, to suggest new investigations; and *(iii)* to gather structure and trustworthy knowledge to the proper alignment of new research activities.

Figure 2 illustrates the SLR process in its three different steps. According to Kitchenham et al., an SLR process must be seen as a three-fold process [Kitchenham 2004]:

— Planning: in this step of the process, researchers must conduct a planning activity in which some key factors should be clarified: research questions and a detailed review protocol are produced. The review protocol should establish the source of information for studies, define search strings, and criteria for inclusion, exclusion, and quality of studies;
— Conducting the review: once the review protocol is done, researchers are allowed to start conducting the review. This step of the process aims to perform the following tasks: to carry out the search on the selected source of studies, select studies based on the inclusion/exclusion criteria, evaluate the quality of the studies, and, finally, perform the data extraction;
— Documenting the review: this final step addresses the interpretation and reporting of the results of the systematic review, delivering its results for potential interested. Thus, researchers could accomplish descriptive statistics, demographic information, visual information, and several discussion.

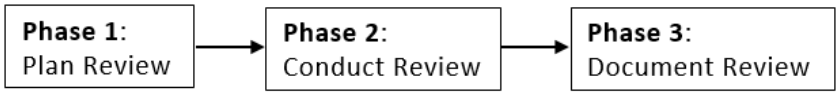


Fig. 2. The SLR three-fold process [Brereton et al. 2007].

In our study, we have conducted an SLR whose goals are to identify automated approaches to identify XBIs. Next, besides some planning and conducting issues, we present a compilation of our SLR's results.

3.1. Planning

3.1.1. Objectives. This research goal is to identify actual approaches for automated detection of XBIs, highlighting their strengths and weakness. In addition to that, this research's results are supposed to support a wide analysis over the evolution of automated XBI approaches, providing guidelines for future research efforts towards contributing to this research topic.

3.1.2. Research Questions. According to [Brereton et al. 2007], Research questions are the foundation to define search strings useful to perform searches in scientific-indexed database of studies. Research questions are extremely important to define the research scope, guiding the researcher on how to collect and analyse the studies included in the SLR. Research questions are defined in the review protocol and researchers can refine them after conducting some pilot research to calibrate the search string.

Regarding the particular purpose of identifying automated strategies of XBI, we defined two Research Questions (RQ):

— *[RQ1] What are the existent automated approaches to detect Cross-Browser Incompatibilities (XBIs)?*
— *[RQ2] What are the advantages and disadvantages on using the identified approaches?*
—

3.1.3. Source Search Method. The research method we adopted consists of running a generic search string in different scientific-indexed database available online. Next we present the generic search string we defined:

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

We performed our search in the three different scientific database: ACM Digital Library[7], IEEE Xplore[8], and Springerlink[9]. We have calibrated the search string to each library according to the particular rules of their search engine.

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

— Inclusion criterion 1 (IC1): all studies addressing the problem of automated detection of XBIs were included in the SLR (*Generation 3 - Crawl and capture approaches*);
— Exclusion criterion 1 (EC1): all studies addressing approaches for preventing XBI during the coding phase of the development process were excluded;
— Exclusion criteria 2 (EC2): all repeated entry identified in more than one database were excluded; and
— Exclusion criteria 3 (EC3): all studies reporting approaches which do not mention XBI detection were excluded.

3.1.5. Quality assessment criteria

— Quality assessment criteria 1 (QA1): Publications written in English; and
— Quality assessment criteria 2 (QA2): Publications from peer-reviewed conferences and journals.

Regarding these two aforementioned quality aspects, studies that not achieve the required quality must be excluded of the SLR. However, all of the studies identified were written in English and were published in peer-reviewed venues.

3.2. Conducting the review

Aiming at identifying studies to compose our SLR, we have adapted our generic search string to load properly in the search engine of the selected databases. We have conducted a two-step search: our first search was loaded from 05/15/2015 to 09/12/2015, after that, our search was updated from 08/15/2016 to 08/29/2016.

Regarding the selection process, we applied our inclusion/exclusion/quality criteria in all of the 953 studies identified during the search process.

A Tabela 3.2 apresenta o nmero de estudos retornados em cada biblioteca digital.

Table I. Number of papers identified in the initial search per digital library.

Digital Library	Number of articles
ACM Portal	167
IEEEXplore	183
Springer	603
Total	953

These criteria were initially applied in the paper's title and abstract, then resulting in the identification of 17 out of 953 studies. In order to make the selection process

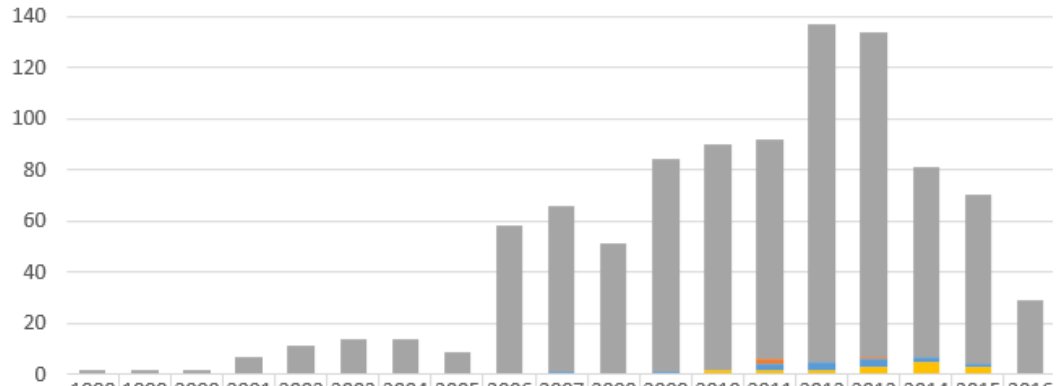
[7]see: http://dl.acm.org/
[8]see: http://ieeexplore.ieee.org
[9]see: http://link.springer.com/

more trustworthy, including the inclusion/exclusion criteria application, all of the judgments were performed by one author and reviewed by another.

3.3. Reporting the review

Figure 3 presents a bar graph including the number of studies analyzed by the first step our SLR. The studies are represented in a year-by-year distribution regarding their year of publication. The studies were published between 1998 and 2016 – in a 19-year-period. In addition to that, the publication number has been increased after 2006, reflecting the popularization of the web applications.

After applying a classification based on the IC/ECs, 96% out of the 953 were excluded by EC3 criterion. During the selection process, we noticed that several studies mention the term cross-browser, however they do not deal with the description of automated techniques for detection XBIs. In the bottom of Figure 3, it is possible to observe that the number of papers that were excluded of the SLR based on each exclusion criterion.



	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016
EC3	2	2	2	7	11	14	14	9	58	65	51	83	88	86	132	127	74	66	29
EC2	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	1	0	0	0
EC1	0	0	0	0	0	0	0	0	0	1	0	1	0	2	3	3	2	1	0
IC1	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	3	5	3	0
Total	2	2	2	7	11	14	14	9	58	66	51	84	90	92	137	134	81	70	29

Fig. 3. Number of papers identified from 1998 to 2016 and classified by their inclusion and exclusion criteria.

Figure 3 also presents a bar-plot with the distribution of the included papers regarding their year of publication. Analyzing the 17 studies included in our SLR, it is possible to notice that the first studies on automated approaches for XBIs were published in 2010. In addition to that, our study highlights that most of the studies included were published in 2014.

Regarding the numbers, 65% out of 17 papers were published in the last three years and 88% were published in the last 5 years. This shows how XBI approaches have been constantly investigated by researchers in the last 6 years. We noticed that 24 researchers authored or co-authored the selected studies. S. Choudhary is involved in 35% of all publications. Further, R. Orso (29% out of 17 publications) and M. Prasad (23% out of 17 publications) were the authors with the highest number of publications after S. Choudhary. Regarding the collaboration among the aforementioned authors,

they published 3 of 17 papers together and 41% out of 17 studies individually or co-authoring other researchers. Table 3.3 presents the number of publications per authors and their affiliation.

Table II. Researchers and the number of publications.

Autores	Publicaes	Organizao/Pas
S. Choudhary	6	Georgia Institute of Technology, USA
A. Orso	5	Georgia Institute of Technology, USA
M. Prasad	4	Group Fujitsu Laboratories of America, USA
M. Burger	3	Saarland University, Germany
V. Dallmeier	3	Saarland University, Germany
A. Zeller	3	Saarland University, Germany
H. Zeng	3	School of Computer Engineering and Science, China
H. Versee	2	Georgia Institute of Technology
M. Dumas	2	University of Tartu, Estonia
T. Orth	2	Saarland University, Germany
T. Saar	2	Software Technology and Applications Competence Center, Estonia
N. Semenenko	2	University of Tartu, Estonia
M. Kaljude	1	Software Technology and Applications Competence Center, Estonia
A. Mesbah	1	University of British Columbia Vancouver, Canada
M. Mirold	1	Testfabrik, German
B. Pohl	1	Testfabrik, German
E. Selay	1	University of Wollongong, Australia
Z. Zhou	1	University of Wollongong, Australia
L. Sanchez	1	Fundao Educacional Inaciana, Brazil
P. Aquino	1	Instituto de Pesquisas Tecnolgicas, Brazil
J. Zou	1	University of Petroleum, China
X. Li	1	School of Computer Engineering and Science, China
S. Xu	1	School of Computer Engineering and Science, China
H. Shi	1	School of Computer Engineering and Science, China

Next section presents a detailed analysis on the automated approaches identified in our SLR.

4. RESULTS

In this section, we present the results of the SLR conducted to extract the different approaches used in the academia to automatically detect XBIs. All of the studies in the SLR report XBIs as client-side components of a web application which do not support cross-browser features.

It is worth noticing that the 17 selected articles in the SLR reported 8 different tools for detecting XBIs. Henceforth, there are multiple articles which report different experiments and evaluations conducted in the same tool and there are also studies that, even though described technological solutions for the identification of XBI, did not implement or named a tool. The tools which were identified in the studies are presented in Table III.

Figure 4 illustrates the number of articles which address only Functionality XBIs (2 articles), only Layout XBIs (8 articles) and both (7 articles).

XBI detection strategies evolved through time, targeting Layout and Functionality XBIs. Multiple approaches have been reported with the goal of improving the precision of XBI detection. Web applications consist of multiple web pages and multiple functionalities are built within them. Henceforth, the studies identified in the literature which lead to improved precision while identifying XBIs leads towards decreasing software development costs.

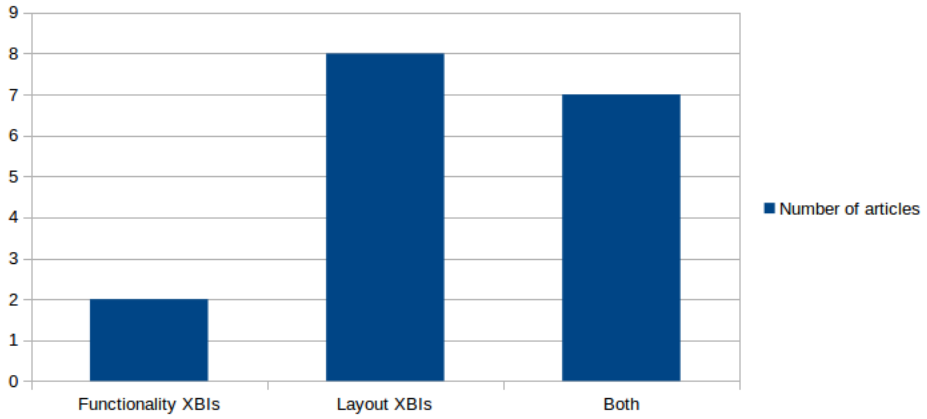


Fig. 4. Number of articles which describe XBI detection strategies which detect Functionality XBIs, Layout XBIs or both.

In regards to the validation conducted in the selected articles, Figure 5 illustrates the number of studies which do not mention empirical validation (5 articles), which conducted a Case Study (4 articles) and which conducted an Experiment (8 articles). In this classification, only studies which investigated the effectiveness of XBI detection approaches were considered in the Case Study and Experiment categories. Studies which addressed validation methodologies measuring code coverage [Dallmeier et al. 2012] and performance metrics [Selay et al. 2014] were not considered, since they do not represent validation processes which measure the effectiveness of XBI detection algorithms.

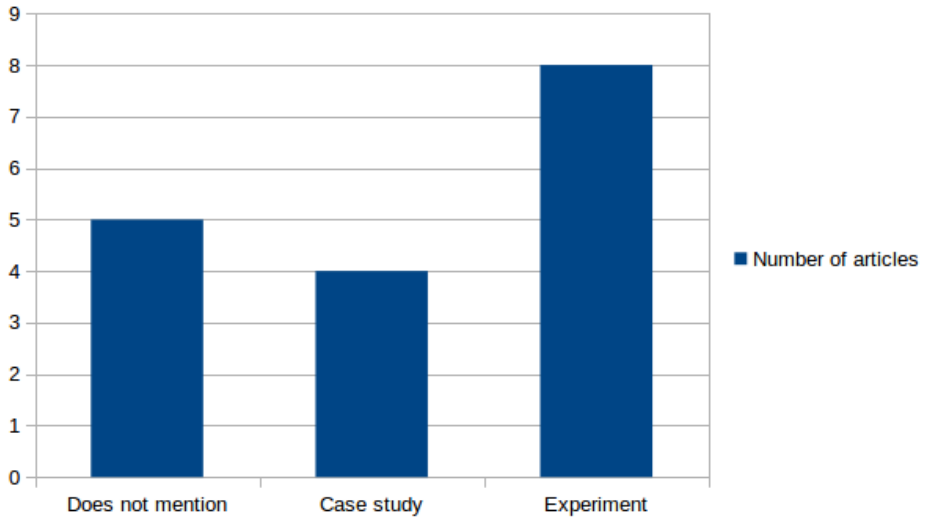


Fig. 5. Number of articles which describe XBI detection strategies and validation process conducted.

The experiments identified in the studies were conducted considering 5 distinct XBI detection tools: WebDiff [Choudhary et al. 2010b], CrossCheck [Choudhary et al.

2012], X-Pert [Choudhary et al. 2013], BrowserBite [Semenenko et al. 2013] and ADDII [Sanchez and Jr. 2015]. Figure 6 illustrates the number of websites considered in each experiment:

— WebDiff - 9 websites.
— CrossCheck - 7 websites.
— X-Pert - 14 websites.
— BrowserBite - 140 websites.
— ADDII - 10 websites.

Even though each experiment was conducted measuring the effectiveness of the XBI detection strategies proposed, it is not possible to compare the results among the experiments, since they were conducted with different websites. However, Choudhary et al. compare the results of CrossCheck and X-Pert, whereas X-Pert present a higher precision than the other tool in the same group of websites [Choudhary et al. 2013].

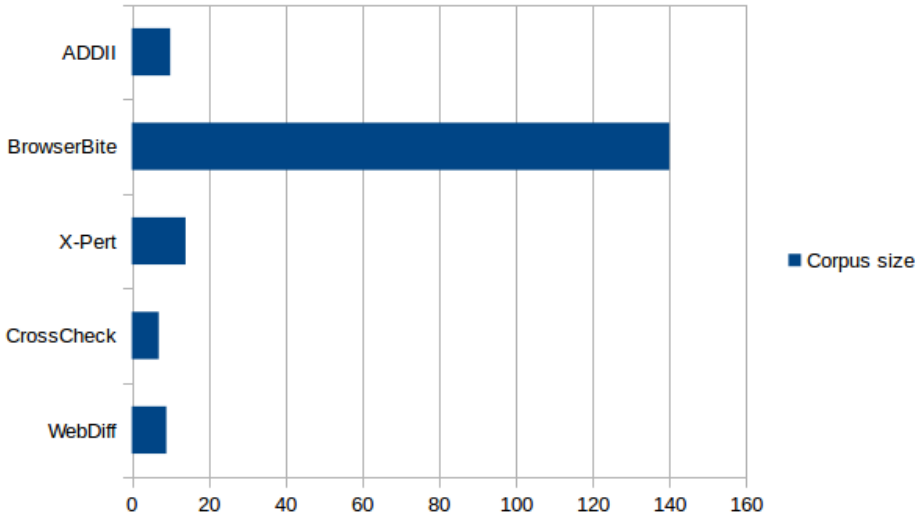


Fig. 6. Number of websites considered in the experiments conducted for validating each approach.

Figure 7 illustrates the number of articles which used each browser in their experiments: Mozilla Firefox (5 articles), Google Chrome (3 articles) and Microsoft Internet Explorer (5 articles). Other browsers such as Opera, Konqueror, Microsoft Edge were not reported in any article.

Each study selected in this SLR proposes different approaches which might focus on one specific group of XBI. The technological approaches identified in this review were: **Structural DOM analysis**, **Screenshot comparison**, **Isomorphism in graphs**, **Machine learning**, **Relative layout**, **Adaptive Random Tests,** and **Static Analysis**. Table III lists the selected studies of the SLR process and shows which approaches have been implemented by each studie with its corresponding tool. Table III also shows the number of studies which explored each technological approach for detecting XBIs.

Figure 8 illustrates the number of articles which used each technological approach.

Next section presents the evolution and a high level description of the XBI detection approaches used in the selected articles.

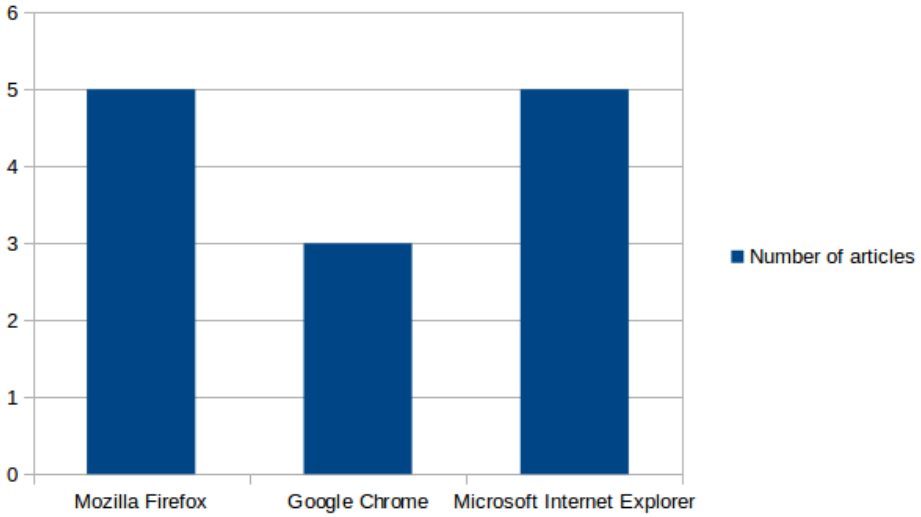


Fig. 7. The browsers used in the experiments reported in the SLR.

Table III. Selected articles in the SLR, the tools which were implemented in each article and the technological approaches which were used to detect XBIs: **Structural DOM analysis - DOM**, **Screenshot comparison - SC**, **Isomorphism in graphs - IG**, **Machine learning - ML**, **Relative layout - RL**, **Adaptive Random Tests - ART** and **Static Analysis - SA**.

#	Paper	DOM	SC	IG	ML	RL	ART	SA	Tool
1	WebDiff: Automated identification of cross-browser issues in web applications [Choudhary et al. 2010b]	X	X	-	-	-	-	-	WebDiff
2	A cross-browser web application testing tool [Choudhary et al. 2010a]	X	X	-	-	-	-	-	WebDiff
3	Detecting cross-browser issues in web applications [Choudhary 2011]	X	X	-	-	-	-	-	WebDiff
4	Automated cross-browser compatibility testing [Mesbah and Prasad 2011]	X	-	X	-	-	-	-	CrossT
5	CrossCheck: Combining crawling and differencing to better detect cross-browser incompatibilities in web applications [Choudhary et al. 2012]	X	X	X	X	-	-	-	CrossCheck
6	WebMate: A tool for testing web 2.0, applications [Dallmeier et al. 2012]	X	-	X	-	X	-	-	WebMate
7	X-Pert: Accurate identification of cross-browser issues in web applications [Choudhary et al. 2013]	X	X	X	X	X	-	-	X-Pert
8	Browserbite: Accurate cross-browser testing, via machine learning over image features [Semenenko et al. 2013]	-	X	-	X	-	-	-	Browserbite
9	WebMate: web application test generation in the real world [Dallmeier et al. 2014]	X	-	X	-	X	-	-	WebMate
10	Modeling web application for cross-browser compatibility testing [Li and Zeng 2014]	X	-	X	-	-	-	-	-
11	X-Pert: A web application testing tool for, cross-browser inconsistency detection [Choudhary et al. 2014]	X	X	X	X	X	-	-	X-Pert
12	Adaptive random testing for comparison, in regression web testing [Selay et al. 2014]	-	X	-	-	-	X	-	-
13	WebMate generating test cases, for web 2.0 [Dallmeier et al. 2013]	X	-	X	-	X	-	-	WebMate
14	Cross-browser testing in Browserbite [Saar et al. 2014]	-	X	-	X	-	-	-	Browserbite
15	Automatic deformations detection in internet interfaces: ADDII [Sanchez and Jr. 2015]	-	X	-	-	-	-	-	ADDI
16	Static Analysis Technique of Cross-Browser Compatibility Detecting [Xu and Zeng 2015]	-	-	-	-	-	-	X	-
17	Cross-browser compatibility testing based on model comparison [Shi and Zeng 2015]	X	-	X	-	-	-	-	-
-	Total	12	10	9	5	5	1	1	7

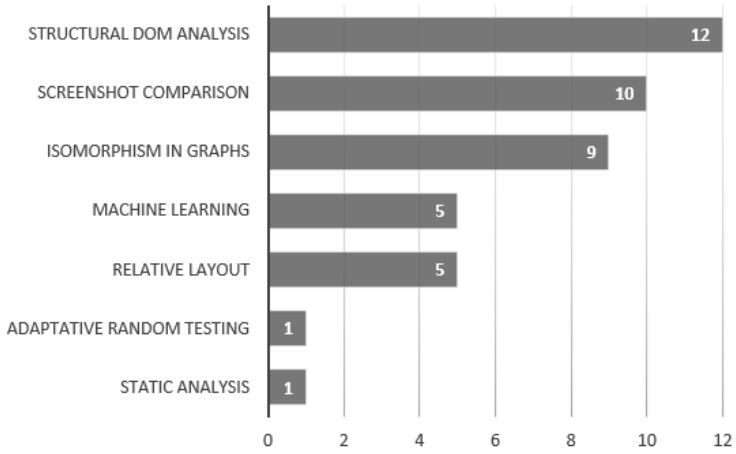


Fig. 8. Number of articles which implement each technological approach to detect XBIs.

4.1. Evolution of the state of art

XBI detection techniques evolved through time employing a variety of techniques. Many techniques were influenced by previous work and focused the identification of distinct classes of XBI: Layout and Functionality XBIs. Figure 9 illustrates a timeline view of the articles identified in the review, the explicit relationship of research approaches and the classes of XBI which are addressed in the technique.

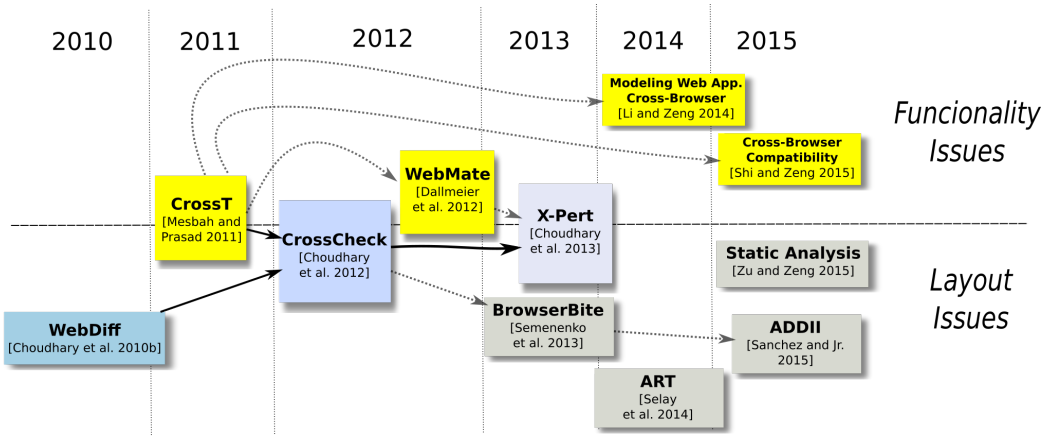


Fig. 9. Timeline of the XBI detection strategies identified in the review and the class of XBI issues which are addressed by each of them.

The first approach identified in the review for detecting XBI was reported in the WebDiff tool [Choudhary et al. 2010b; 2010a; Choudhary 2011]. WebDiff focused on Layout XBIs identification and segmented a web application using its DOM structure. In WebDiff's approach, the DOM structure of a web application was rendered in different browsers. The same DOM element representation in the multiple browsers was compared to one another using their DOM attributes and screenshot comparison.

According to the XBI detection approach classification scheme defined in this paper, WebDiff used two approaches: **Structural DOM Analysis** and **Screenshot Comparison**.

In 2011, Mesbah and Prasad reported the CrossT tool [Mesbah and Prasad 2011]. CrossT introduced an approach based on Graphs Isomorphism for detecting functionality XBI. CrossT modeled a web application as a finite state machine in which states represented the current DOM structure of the web application and the transitions represented click events dispatched in an specific element. This finite state machine graph illustrated changes made to the DOM structure after click events were dispatched in the web application. CrossT generated multiple finite state machine graphs of a single web application rendered in different browsers. Then, the multiple finite state machine graphs (one for each browser) were compared and differences identified between the graphs were reported as XBIs. CrossT also reported the use of DOM attributes to compare DOM elements rendered in multiple browsers. CrossT addressed the identification of Functionality and Layout XBIs and used two approaches: **Structural DOM Analysis** and **Isomorphism in Graphs**.

After CrossT publication, Choudhary et al. implemented a new XBI detection tool titled CrossCheck [Choudhary et al. 2012]. CrossCheck was implemented as an extension of WebDiff [Choudhary et al. 2010b; 2010a; Choudhary 2011] which incorporated the Functionality XBI detection approach used in CrossT [Mesbah and Prasad 2011]. CrossCheck implemented the same DOM element segmentation, screenshot and DOM attributes comparison mechanisms of WebDiff to identify Layout XBIs, while complementary modeling finite state machine graph, similarly to CrossT, to identify Functionality XBIs. Furthermore, CrossCheck also implemented machine learning in the XBI detection mechanism. Choudhary et al. reported that the XBI detection algorithm should not rely on absolute comparison of DOM attributes and screenshot comparison to identify XBIs. Web developers, while searching for XBIs in a web application, present a more strict XBI searching strategy depending on the browser they are analysing. For instance, web developers might report a XBI in a more modern and updated browser, while if the same characteristics were observed in an older browser they would not imply in a XBI. Henceforth, CrossCheck reported the use of machine learning to automatically adjust the threshold for XBI detection for each browser, in order to improve the false positive rate of the XBI detection algorithm. Finally, CrossCheck according to the SLR classification used the following approaches: **Structural DOM Analysis**, **Screenshot Comparison**, **Isomorphism in Graphs** and **Machine Learning**.

Still in 2012, Dallmeier et al. published an article on the WebMate tool [Dallmeier et al. 2012; 2013; Dallmeier et al. 2014]. WebMate worked similarly to CrossT [Mesbah and Prasad 2011], modeling the web application as a finite state machine to identify Functionality and Layout XBIs. WebMate's finite state machine approach was inspired by GUI testing studies with focus on automatic test case generation. WebMate's publication was also the first to mention the use of relative positioning features to detect XBIs. While segmenting a web application according to its DOM elements, if a specific element is misplaced in the application all its children would also be classified as misplaced. Therefore, Dallmeier et al. describe the use of relative position of web elements to improve the accuracy of the XBI detection approach, reducing the number of false positive XBI identification [Dallmeier et al. 2012]. Dallmeier et al. WebMate used the following approaches in their XBI detection algorithm: **Structural DOM Analysis**, **Isomorphism in Graphs** and **Relative layout**.

In 2013, two XBI detection tools were published: the X-Pert [Choudhary et al. 2013; 2014] and the Browserbite [Semenenko et al. 2013; Saar et al. 2014]. X-Pert extended the CrossCheck tool, focusing Functionality and Layout XBI using the same

approaches used in CrossCheck [Choudhary et al. 2013; 2014]. Nevertheless, X-Pert refined the approaches used in CrossCheck and included the relative positioning concept applied in WebMate [Dallmeier et al. 2012]. Thus, X-Pert used the following approaches according to the XBI detection approach classification of the SLR: **Structural DOM Analysis**, **Screenshot Comparison**, **Isomorphism in Graphs**, **Machine Learning** and **Relative layout**.

The BrowserBite tool was implemented in 2013 and is now online available[10] [Semenenko et al. 2013; Saar et al. 2014]. BrowserBite, differently from its predecessor XBI detection approaches, focuses the identification of Layout XBIs and segments a web application through its complete screenshot in regions of interest, based on borders and color changes. After collecting web applications screenshots as rendered in multiple browsers, each browser screenshot is segmented in regions of interest. Then, the same region of interest rendered in multiple browsers are matched and compared to one another to identify XBIs. The comparison is realized using machine learning techniques and a set of features extracted from the region of interest image. BrowserBite used the following approaches according to the classification proposed in the SLR: **Screenshot Comparison** and **Machine Learning**.

By 2014 and 2015, three other XBI detection approaches were reported in the literature using similar strategies identified in previous research:

— In [Li and Zeng 2014; Shi and Zeng 2015], two separate XBI detection approaches were reported following the same approach used in CrossT [Mesbah and Prasad 2011]. Both approaches modeled the user interaction with the web application, similarly to Mesbah's and Prasad's finite state machine modeling strategy. The recent approaches, on the other hand, differ in the type of events which are considered while building the interaction model and AJAX. Both approaches used **Structural DOM Analysis** and **Isomorphism in Graphs**.
— In [Sanchez and Jr. 2015], the ADDII tool was proposed, using an image comparison approach similar to the one proposed in the BrowserBite tool [Semenenko et al. 2013; Saar et al. 2014]. The ADDI tool, on the other hand, do not segment images, comparing full screenshots as rendered in different browsers to identify XBIs. The ADDII tool according to the classification proposed in the SLR used solely the **Screenshot Comparison** approach.

Finally, concurrently to other studies, two distinct approaches were reported for identifying XBIs:

— **Adaptive Random Testing** for improving image similarity algorithms performance [Selay et al. 2014].
— **Static Analysis** which consists of building a repository of web applications cross-platform features and tracking the use of these features in web applications [Xu and Zeng 2015].

Next section presents a low level description of the XBI detection approaches reported in the state of art.

4.2. XBI detection approaches

The next sub-sections present the different approaches used to identify XBIs. The approaches are chronologically ordered by when the technological solution was first proposed in the studies.

[10] http://browserbite.com/

4.2.1. Structural DOM Analysis. The DOM (*Document Object Model*) defines a logic structure for HTML and XML documents to allow programs and scripts to dynamically access and update the content[11]. The DOM represents HTML and XML documents as a tree data structure and browsers frequently provide an API(Application Program Interface) to be used by web applications.

Almost all studies selected in the SLR process used the DOM structure of a webpage to detect XBIs. These studies approaches for detecting XBI consisted of comparing attributes and elements of DOM representations rendered in different browsers.

Initially, multiple DOM representations of a single webpage rendered in different browsers were captured. Then, these DOM representations were compared to one another for similarity, in order to identify attributes and elements which did not present the same value [Choudhary et al. 2010b; 2010a; Choudhary 2011; Mesbah and Prasad 2011; Choudhary et al. 2012; Dallmeier et al. 2012; Choudhary et al. 2013; Dallmeier et al. 2013; Dallmeier et al. 2014; Li and Zeng 2014; Choudhary et al. 2014; Shi and Zeng 2015]. In these approaches, if the same element in different DOM representations presented incompatible attribute values, then this element would be possibly marked as an XBI which should be later reported to developers. Most studies used a selected group of attributes verifications in this step.

The first article to use this approach was Choudhary et al. [Choudhary et al. 2010b], and the same approach was further developed in other articles, such as [Choudhary et al. 2012; 2013; Mesbah and Prasad 2011]. While the studies evolved through time, it was observed that simply comparing the attributes in the DOM structure as rendered in 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 considering not only comparing DOM representations but using other strategies such as attribute value normalization, machine learning, among other techniques which are described in the next sections of the paper.

Still considering structural DOM analysis, the studies [Choudhary et al. 2010b; Choudhary et al. 2012; 2013] presented implementation detailing how to match elements from different DOM representations. DOM representations rendered in multiple browsers might present structural differences which might difficult the task of comparing attributes and elements which can be miss placed depending on the browser in which it was rendered. Henceforth, these studies presented a technique based on using attributes such as *tagName*, *id*, *xpath* and a hash mapping of the inner content (*innerHTML*) to match similar elements from different DOM representations.

4.2.2. Screenshot Comparison. DOM structure information can be used to assist the process of identifying XBI on webpages. However, DOM does not present information about each element exact appearance on the screen [Choudhary et al. 2010b]. Thus, in regards to the identification of **Layout issues** XBIs, many studies reported the use of image comparison techniques in screenshots captured from the web application [Choudhary et al. 2010b; 2010a; Choudhary 2011; Choudhary et al. 2012; 2013; Semenenko et al. 2013; Choudhary et al. 2014; Selay et al. 2014; Saar et al. 2014; Sanchez and Jr. 2015].

The primary method for comparing images is a direct comparison of histograms. However, using a rigid comparisson of histograms with no regards to global features (such as symmetry, kurtosis and number of peaks) to identify XBIs might raise many false positives [Saar et al. 2014]. In this context, 5 main image comparison techniques were used in the studies: Earth Movers' Distance (EMD) [Choudhary et al. 2010b;

[11] see: https://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/

2010a; Choudhary 2011], $\chi 2$ Distance [Choudhary et al. 2012; 2013; 2014], Image Segmentation and correlation-based image comparison [Semenenko et al. 2013; Saar et al. 2014], pHash and Perceptual Image Diff [Sanchez and Jr. 2015].

The EMD is based on a solution to transportation problems from linear optimization and as an image comparisson technique is more robust than histogram matching techniques [Rubner et al. 2000]. EMD is a distance metric which can be applied to color or texture distributions, for example, given the space of points can be equipped with a similarity metric [Levina and Bickel 2001]. Choudhary et al. reported the use of EMD to individually compare screenshots taken in different browsers for each element of the DOM structure of a webpage [Choudhary et al. 2010b; 2010a; Choudhary 2011]. In these studies, Choudhary et al. did not use a single threshold to determine whether two element screenshots were different or not [Choudhary et al. 2010b]. The threshold was set considering the size and the color density of each screenshot image.

The $\chi 2$ distance image comparison technique is adapted from the statistics $\chi 2$ hypothesis test which is used to compare the dispersion between two nominal variables, evaluating the correlation between qualitative variables. The basic principle of the $\chi 2$ hypothesis test is to compare proportion rates with the goal of identifying differences in the frequency an event was detected in two groups. Thus, two groups are classified as similar if the differences in the observed frequency and the expected frequency for multiple categories are very small or close to zero. The $\chi 2$ distance method, as an image processing technique, was successfully used to classify objects, texture and duplicate images [Pele and Werman 2010]. The studies [Choudhary et al. 2012; 2013; 2014] used the $\chi 2$ distance method to compare screenshots of elements of the webpage. Initially, in [Choudhary et al. 2013; 2014], the comparison was used to compare all elements of a webpage. Then, in a latter study [Choudhary et al. 2012], the $\chi 2$ distance method was used to compare only leaf nodes of the DOM tree structure of a webpage. This strategy of comparing only the screenshot of leaf nodes was used to reduce false positives XBI evaluations.

Choudhary et al. also reported that the $\chi 2$ is less expensive computationally and present better precision when comparing with the EMD method [Choudhary et al. 2013; 2014].

The studies [Semenenko et al. 2013; Saar et al. 2014] reported segmenting screenshots rendered in different browsers in small rectangles identified as Region Of Interest (ROI) based on borders and color changes observed in the image. Then, the image comparison was made between the same ROI captured in different browsers. These studies mention the use of a correlation-based image comparison technique [Semenenko et al. 2013].

Sanchez and Aquino Jr. used two different approaches to compare screenshots in the webpage: the pHash[12] and Perceptual Image Diff[13] [Sanchez and Jr. 2015]. Both algorithms were used complementary, whereas pHash was used to report if the screenshots were similar and the Perceptual Image Diff was used to present the amount of pixels which are different. The pHash algorithm is an implementation of the Perceptual Hash concept which is defined as a fingerprint hash of a multimedia file derived from various features from its content[14]. The Perceptual Image Diff, on the other hand, is an algorithm which identifies every pixel in two images which are different. First, the pHash was applied to the images of screenshots captured in different browsers to generate a Hash code for each image. Moreover, the Hash code of each image was compared using the Hamming distance algorithm. The result of the Hamming distance

[12]see: http://pastebin.com/Pj9d8jt5
[13]see: http://pdiff.sourceforge.net
[14]see: http://www.phash.org/

algorithm was used alongside the Perceptual Image Diff method to evaluate if an XBI was identified or not.

4.2.3. Isomorphism in Graphs. The studies [Mesbah and Prasad 2011; Choudhary et al. 2012; Dallmeier et al. 2012; Dallmeier et al. 2014; Choudhary et al. 2013; Dallmeier et al. 2013; Choudhary et al. 2014; Li and Zeng 2014; Shi and Zeng 2015] reported the use of a separate approach which focused the identification of **Functionality** XBIs. These approaches consisted of simulating user interaction scenarios with the application, mainly click events in elements of the webpage, to trigger changes in the DOM representation of the web application in one specific browser. Then, the same web application would be subject to the same simulated user interaction scenarios in another browser to compare if the same DOM representations were obtained as in the first browser. In order to conduct this evaluation, these studies mapped the DOM representations and click events simulations into a graph representation and, then, used an Isomorphism Graph algorithm to identify functionalities which were not available depending on the browser the user was using.

The first study to use this approach was [Mesbah and Prasad 2011]. This study extended an Ajax applications crawler tool, named Crawljax [Mesbah and van Deursen 2009; Mesbah et al. 2012]. Crawljax related studies propose the use of event simulation in elements of a webpage, such as click events, to capture changes in the DOM structure. Ajax web applications present a dynamic DOM structure which can be changed during user interaction. Henceforth, in order to be able to map these changes, the crawler is supposed to generate events on the webpage elements to trigger these changes and potentially access information which was not available as the webpage DOM structure was initially loaded in the browser.

Mesbah and Prasad used the Crawljax to trigger DOM structure changes, given that click events were dispatched to the webpage and mapped the generated data in a finite state machine, in which the states represents the multiple generated DOM structures and the transitions represented user actions (the clicks which triggered each DOM structure change). The top level representation of this finite state machine is a graph representation and it was named Navigation model in this study. Their approach would map multiple Navigation models, one for each browser. Then, the Navigation models rendered in different browsers would be compared using an equivalence check procedure, specifically running an Graph Isomorphism algorithm [Mesbah and Prasad 2011].

In regards to the equivalence check procedure, the problem of Graphs Isomorphisms consists in identifying a complete structure of a graph inside another graph [Buchanan et al. 1984]. Mesbah and Prasad run a Graph Isomorphism algorithm and all incompatibilities identified by this algorithm were reported as **Functional** XBIs, representing click functionalities which were observed in one browser but not another [Mesbah and Prasad 2011].

After Mesbah and Prasad study, other XBI detection approaches implemented similar strategies, such as [Choudhary et al. 2012; Dallmeier et al. 2012; Dallmeier et al. 2014; Choudhary et al. 2013; Dallmeier et al. 2013; Choudhary et al. 2014].

4.2.4. Machine Learning. Machine learning is an area of Artificial Intelligence which has the objective of developing computational techniques which are capable of automatically acquiring knowledge on how to perform an specific task without being explicitly programmed to do so. Weiss and Kulikowski define machine learning as a computer program which takes decisions based on experiences acquired and accumulated through successfully previous solutions to a problem [Weiss and Kulikowski 1991]. A frequent problem faced by research studies on XBI detection solutions was the high number of false positives identified by their techniques. In regards to that, a group of

researches investigated the use of machine learning techniques to reduce the number of false positives in their XBI detection solutions.

Machine learning techniques are divided in three main categories: supervised learning, unsupervised learning and reinforcement learning. All XBI detection studies which were selected in this SLR used supervised learning techniques. In supervised learning, concepts are induced into the system from pre-classified examples. The XBI detection approaches which used machine learning, used examples of HTML/CSS/JS website codes which contained XBIs and examples of codes which did not. These labeled examples were used as learning data to distinct machine learning techniques. The uses of machine learning among XBI detection studies varied which characteristics of a web application were used as features in the learning phase and which machine learning technique was used.

The first studies which used machine learning techniques for XBI detection were [Choudhary et al. 2012; 2013; 2014]. 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 [Quinlan 1993]. These articles used a decision tree to classify every element of the DOM structure rendered in different browsers represented a XBI or not. The decision tree, in these articles, was trained considering the following features:

(1) **Size difference ratio**: differences in the size of the same elements rendered in different browsers;
(2) **Displacement**: the euclidean distance between the position of the same elements rendered in different browsers;
(3) **Area**: the smaller area of the same elements rendered in different browsers;
(4) **Leaf Node Text Differences**: a boolean value identifying whether there are differences between the texts of the same elements rendered in different browsers; and
(5) **Image distance**: the $\chi 2$ distance screenshot comparison of the same elements rendered in different browsers.

After [Choudhary et al. 2012; 2013; 2014], Semenenko et al. and Saar et al. reported the use of decision trees and of neural networks to detect XBIs in web applications [Semenenko et al. 2013; Saar et al. 2014]. Neural networks stand for a mathematical model inspired in the neural structure which acquire knowledge through experience. The most essential characteristic of neural networks is related to their capability of learning and improving how they perform a specific task through the continuous training [Heaton 2008]. Semenenko et al. and Saar et al. conducted an experiment to compare how both machine learning approaches performed in a group of website.

It is worth noticing that Semenenko et al. and Saar et al., in order to identify XBI, used region of interests to identify XBIs [Semenenko et al. 2013; Saar et al. 2014], differently from other approaches which used DOM elements of the webpage. Therefore, their neural network approach was executed for each region of interest rendered in two browsers using the following 17 features:

— 10 histograms from screenshots of regions of interest of a webpage rendered in two different browsers;
— Correlation-based image comparison between both screenshots of the same region of interest rendered in two browsers;
— Horizontal and vertical position of the region of interest rendered in two browsers;
— Horizontal and vertical size of the region of interest rendered in two browsers;
— Configuration index indicating the browsers which were used to render the region of interests; and

— Mismatch density metric, which is calculated as a coeficient of the number of region of interests which presented differences in the correlation-based image comparison approach and the total number of region of interests of the webpage.

Both decision trees and neural network machine learning techniques were used to reduce the number of false positive XBI identification. Choudhary et al. reported that the use decision trees improved the results provided by their XBI detection approach . In Semenenko et al. studies, neural network performed better in comparison to decision trees [Semenenko et al. 2013; Saar et al. 2014]. However, it is difficult to compare [Choudhary et al. 2012; 2013; 2014] with [Semenenko et al. 2013; Saar et al. 2014], since these studies used a different set of features and training set.

4.2.5. Relative Layout. A few of the previously mentioned techniques used the position of elements and regions of interest to detect XBIs. In regards to this, Dallmeier et al. proposed that comparison between elements considering their positioning should be conducted using their relative position [Dallmeier et al. 2012; 2013; Dallmeier et al. 2014]. HTML webpages and elements present an hierarchy structure and layout rendering, hence if a parent element is miss aligned in a webpage, its child elements will also present the same miss aligned absolute positioning. Thus, XBI detection approaches which uses absolute position to identify XBIs, might report that the parent and all child elements represent XBIs, however only the parent element was actually miss aligned.

Choudhary et al. also explored this relative layout position evaluation strategy [Choudhary et al. 2013; 2014]. In order to use relative position in the XBI detection approach, Choudhary et al. formally modeled the positioning of elements in a webpage in a graph, titled *Alignment Graph*. The Alignment Graph represents the relationships between parent, child and siblings nodes, represented as elements in the graph, and their relative position in regards to one another, represented as the transitions between each element in the graph. Then, two alignment graphs extracted from two different browsers can be checked for equivalence and any difference observed between the graphs is reported a XBI.

4.2.6. Adaptive Random Testing (ART). Adaptive Random Testing - ART is an extension of the *Random Testing* technique, whereas ART analyses the behavior of test results observed in software and generate test input which are more likely to produce failures in the software [Chen et al. 2010]. Generally, similar test cases tend to identify similar failures, hence ART is a technique which identify test cases which are more likely to improve the coverage of a software input domain.

Selay et al. [Selay et al. 2014] used ART to compare images for equivalence, considering a screenshot comparison approach for identifying XBIs. The ART approach used in [Selay et al. 2014] 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 which explore different parts of the input domain of the software [Chen et al. 2010]. It is worth noticing that, Selay et al. run an experiment with the goal of improving the performance in terms of the time required to identify a differences while comparing two distinct images (screenshots) [Selay et al. 2014].

4.2.7. Static Analysis. Static analysis is a technique which is applied to software artifacts directly, not actually running the program under analysis. Xu et al. [Xu and Zeng 2015] built a HTML5 incompatible features database which consists of elements and attributes that are not supported by three most popular browsers and proposed an algorithm to detect incompatible features in the target web applications based on the static analysis of the HTML source code of the application. The algorithm's input

consists of the code of the target web application and the unsupported HTML5 features by the tested browser. The algorithm's output is a list of XBIs of the target web application. The technique uses regular expression to detect HTML elements of target web applications and identify different HTML features.

5. THREATS TO VALIDITY

The SLR process was conducted in order to identify all automatic XBI detection approaches which were used in the state of art. However, in regards to the internal validity of this SLR, there might be studies which report the use of XBI detection techniques but were not included in the SLR. In order to reduce the risks associated to this specific threat, the SLR process was conducted by two researchers, whereas all studies and the applicability of inclusion/exclusion/quality criteria were discussed and reviewed between them.

Systematic literature studies can be conducted following two ways of searching for studies: snowballing and database searches [Badampudi et al. 2015]. Our SLR process used database searches in ACM, IEEEXplore, and Springerlink to search for articles related to XBI detection techniques. Henceforth, the results of this SLR could be enhanced if other databases were included and snowballing was also applied. However, the inclusion of other studies in the SLR would imply the addition of other approaches related to one of the approaches already described in this SLR or, at most, add a new approach category to the results of this SLR, not invalidating any of the previously reported results. Moreover, this SLR could be updated with other sources and the use of snowballing in future works.

6. FINAL REMARKS

Cross Browser Incompatibilities (XBIs) are frequent in web projects and represent serious problems for web application developers. This study conducts a Systematic Literature Review Process with the goal of identifying the many different approaches which have been used to automatically detect XBIs from the state of art approaches.

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 algorithms, relative layout, ART and static analysis of the code. The classification of the studies technological approaches and the comparison analysis provided between the XBI detection strategies can be used in the elaboration of new strategies and tools.

Future works include the implementation of automatic detection XBIs in different platforms (between desktop and mobile devices) and investigating techniques which assist the developer in the diagnosis and correction of found XBIs.

ACKNOWLEDGMENTS

Omitted due to blind revision

REFERENCES

Deepika Badampudi, Claes Wohlin, and Kai Petersen. 2015. 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)*. ACM, New York, NY, USA, Article 17, 10 pages. DOI:http://dx.doi.org/10.1145/2745802.2745818

Pearl Brereton, Barbara A. Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil. 2007. Lessons from Applying the Systematic Literature Review Process Within the Software Engineering Domain. *Journal of Systems and Software* 80, 4 (April 2007), 571–583. DOI:http://dx.doi.org/10.1016/j.jss.2006.07.009

B. G. Buchanan, E. H. Shortliffe, and W. van Melle. 1984. *EMYCIN: A Knowledge Engineer's Tool for Constructing Rule-Based Expert Systems*. Addison-Wesley, Chapter 15, 303–313. http://people.dbmi.columbia.edu/~ehs7001/Buchanan-Shortliffe-1984/Chapter-15.pdf

David Budgen and Pearl Brereton. 2006. Performing Systematic Literature Reviews in Software Engineering. In *Proc. of the 28th International Conference on Software Engineering (ICSE 2006)*. ACM, New York, NY, USA, 1051–1052. DOI:http://dx.doi.org/10.1145/1134285.1134500

Tsong Yueh Chen, Fei-Ching Kuo, Robert G. Merkel, and T. H. Tse. 2010. Adaptive Random Testing: The ART of Test Case Diversity. *Journal of Systems and Software* 83, 1 (Jan. 2010), 60–66. DOI:http://dx.doi.org/10.1016/j.jss.2009.02.022

S.R. Choudhary. 2011. Detecting cross-browser issues in web applications. In *Proc. of the 33rd International Conference on Software Engineering (ICSE 2011)*. 1146–1148. DOI:http://dx.doi.org/10.1145/1985793.1986024

S.R. Choudhary, M.R. Prasad, and A. Orso. 2012. CrossCheck: Combining Crawling and Differencing to Better Detect Cross-browser Incompatibilities in Web Applications. In *Proc. of the 5thInternational Conference on Software Testing, Verification and Validation (ICST 2012)*. 171–180. DOI:http://dx.doi.org/10.1109/ICST.2012.97

S.R. Choudhary, M.R. Prasad, and A. Orso. 2013. X-PERT: Accurate identification of cross-browser issues in web applications. In *Proc. of the 35th International Conference on Software Engineering (ICSE 2013)*. 702–711. DOI:http://dx.doi.org/10.1109/ICSE.2013.6606616

S.R. Choudhary, H. Versee, and A. Orso. 2010a. A cross-browser web application testing tool. In *Proc. of the 2010 IEEE International Conference on Software Maintenance (ICSM 2010)*. 1–6. DOI:http://dx.doi.org/10.1109/ICSM.2010.5609728

S.R. Choudhary, H. Versee, and A. Orso. 2010b. WEBDIFF: Automated identification of cross-browser issues in web applications. In *Proc. of the 2010 IEEE International Conference on Software Maintenance (ICSM 2010)*. 1–10. DOI:http://dx.doi.org/10.1109/ICSM.2010.5609723

S. R. Choudhary, Mukul R. Prasad, and Alessandro Orso. 2014. 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)*. ACM, New York, NY, USA, 417–420. DOI:http://dx.doi.org/10.1145/2610384.2628057

Valentin Dallmeier, Martin Burger, Tobias Orth, and Andreas Zeller. 2012. WebMate: A Tool for Testing Web 2.0 Applications. In *Proc. of the Workshop on JavaScript Tools (JSTools 2012)*. ACM, New York, NY, USA, 11–15. DOI:http://dx.doi.org/10.1145/2307720.2307722

Valentin Dallmeier, Martin Burger, Tobias Orth, and Andreas Zeller. 2013. *WebMate: Generating Test Cases for Web 2.0*. Springer Berlin Heidelberg, Berlin, Heidelberg, 55–69. DOI:http://dx.doi.org/10.1007/978-3-642-35702-2_5

V. Dallmeier, B. Pohl, M. Burger, M. Mirold, and A. Zeller. 2014. WebMate: Web Application Test Generation in the Real World. In *Proc. of the Workshops of the IEEE 7th International Conference on Software Testing, Verification and Validation Workshops (ICSTW 2014)*. 413–418. DOI:http://dx.doi.org/10.1109/ICSTW.2014.65

Serdar Doan, Aysu Betin-Can, and Vahid Garousi. 2014. Web application testing: A systematic literature review. *Journal of Systems and Software* 91 (2014), 174 – 201. DOI:http://dx.doi.org/10.1016/j.jss.2014.01.010

Cyntrica Eaton and Atif M. Memon. 2007. An Empirical Approach to Evaluating Web Application Compliance Across Diverse Client Platform Configurations. *Int. J. Web Eng. Technol.* 3, 3 (Jan. 2007), 227–253. DOI:http://dx.doi.org/10.1504/IJWET.2007.012055

Vahid Garousi, Ali Mesbah, Aysu Betin-Can, and Shabnam Mirshokraie. 2013. A systematic mapping study of web application testing. *Information and Software Technology* 55, 8 (2013), 1374 – 1396. DOI:http://dx.doi.org/10.1016/j.infsof.2013.02.006

Jeff Heaton. 2008. *Introduction to Neural Networks for Java, 2Nd Edition* (2nd ed.). Heaton Research, Inc.

Barbara Kitchenham. 2004. *Procedures for performing systematic reviews*. Technical Report. Keele University, Keele, UK. 1–26 pages.

E. Levina and P. Bickel. 2001. The Earth Mover's distance is the Mallows distance: some insights from statistics. In *Proc. of the 8th International Conference on Computer Vision (ICCV 2001)*, Vol. 2. 251–256. DOI:http://dx.doi.org/10.1109/ICCV.2001.937632

Xinxin Li and Hongwei Zeng. 2014. 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)*. 1–5. DOI:http://dx.doi.org/10.1109/SNPD.2014.6888745

Ali Mesbah and Mukul R. Prasad. 2011. Automated Cross-browser Compatibility Testing. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE 2011)*. ACM, New York, NY, USA, 561–570. DOI:http://dx.doi.org/10.1145/1985793.1985870

Ali Mesbah and Arie van Deursen. 2009. Invariant-based Automatic Testing of AJAX User Interfaces. In *Proc. of the 31st International Conference on Software Engineering (ICSE 2009)*. IEEE Computer Society, Washington, DC, USA, 210–220. DOI:http://dx.doi.org/10.1109/ICSE.2009.5070522

Ali Mesbah, Arie van Deursen, and Stefan Lenselink. 2012. Crawling Ajax-Based Web Applications Through Dynamic Analysis of User Interface State Changes. *ACM Transactions on the Web* 6, 1, Article 3 (March 2012), 30 pages. DOI:http://dx.doi.org/10.1145/2109205.2109208

Ofir Pele and Michael Werman. 2010. The Quadratic-chi Histogram Distance Family. In *Proc. of the 11th European Conference on Computer Vision: Part II (ECCV 2010) (ECCV'10)*. Springer-Verlag, Berlin, Heidelberg, 749–762. http://dl.acm.org/citation.cfm?id=1888028.1888085

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

Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. 2000. The Earth Mover's Distance As a Metric for Image Retrieval. *International Journal of Computer Vision* 40, 2 (Nov. 2000), 99–121. DOI:http://dx.doi.org/10.1023/A:1026543900054

Tõnis Saar, Marlon Dumas, Marti Kaljuve, and Nataliia Semenenko. 2014. *Cross-Browser Testing in Browserbite*. Springer International Publishing, Cham, 503–506. DOI:http://dx.doi.org/10.1007/978-3-319-08245-5_37

Leandro Sanchez and Plinio Thomaz Aquino Jr. 2015. 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) (Lecture Notes in Computer Science)*, Vol. 9171. Springer International Publishing, 43–53. DOI:http://dx.doi.org/10.1007/978-3-319-21006-3_5

E. Selay, Z. Q. Zhou, and J. Zou. 2014. Adaptive Random Testing for Image Comparison in Regression Web Testing. In *Proc. of the 2014 International Conference on Digital Image Computing: Techniques and Applications (DICTA 2014)*. 1–7. DOI:http://dx.doi.org/10.1109/DICTA.2014.7008093

N. Semenenko, M. Dumas, and T. Saar. 2013. Browserbite: Accurate Cross-Browser Testing via Machine Learning over Image Features. In *Proc. of the 29th IEEE International Conference on Software Maintenance (ICSM 2013)*. 528–531. DOI:http://dx.doi.org/10.1109/ICSM.2013.88

H. Shi and H. Zeng. 2015. Cross-Browser Compatibility Testing Based on Model Comparison. In *Computer Application Technologies (CCATS), 2015 International Conference on*. 103–107. DOI:http://dx.doi.org/10.1109/CCATS.2015.34

Sholom M. Weiss and Casimir A. Kulikowski. 1991. *Computer Systems That Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

S. Xu and H. Zeng. 2015. Static Analysis Technique of Cross-Browser Compatibility Detecting. In *Applied Computing and Information Technology/2nd International Conference on Computational Science and Intelligence (ACIT-CSI), 2015 3rd International Conference on*. 103–107. DOI:http://dx.doi.org/10.1109/ACIT-CSI.2015.26

Received February 2007; revised March 2009; accepted June 2009