# X-PERT: A Web Application Testing Tool for Cross-Browser Inconsistency Detection*

Shauvik Roy Choudhary†, Mukul R. Prasad§, Alessandro Orso†

†College of Computing
Georgia Institute of Technology
Atlanta, GA, USA
{shauvik | orso}@cc.gatech.edu

§Software Systems Innovation Group
Fujitsu Laboratories of America
Sunnyvale, CA, USA
mukul@us.fujitsu.com

## ABSTRACT

Web applications are popular among developers because of the ease of development and deployment through the ubiquitous web browsing platform. However, differences in a web application's execution across different web browsers manifest as Cross-browser Inconsistencies (XBIs), which are a serious concern for web developers. Testing for XBIs manually is a laborious and error-prone process. In this demo we present X-PERT which is a tool to identify XBIs in web applications automatically, without requiring any effort from the developer. X-PERT implements a comprehensive technique to identify XBIs and has been found to be effective in detecting real-world XBIs in our empirical evaluation. The source code of X-PERT and XBI reports from our evaluation are available at `http://gatech.github.io/xpert`.

## Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging

## General Terms

Reliability, Verification

## Keywords

Web Testing, Cross-browser Testing, Layout Testing

## 1 INTRODUCTION

Web applications are increasingly being used for both personal and business activities. Users of such applications might use any web browser to access them and the application is expected to behave consistently across these different environments. However, web applications often exhibit differences when executed in different browsers, leading to Cross-browser Inconsistencies (XBIs). XBIs are discrepancies between a web application's appearance, behavior, or both, when it is run on two different environments. They are not only fairly common but also notoriously difficult to identify and fix. For example, 5328 posts were created and tagged with "cross-browser", on `stackoverflow.com` over the past four years alone. Moreover, nearly 2000 of these posts have been active in the past year [8].

The current practice in industry is to identify such issues through manual inspection of the web application screens across all the different browsers [2]. Such testing is not only human intensive but is also error-prone. If such issues are not identified during testing, they can adversely degrade the experience of the users of the web application with the affected browser. As shown in our evaluation of X-PERT, some XBIs completely disallow users to access the functionality offered by the web application, thereby rendering it useless on the particular platform. Hence, XBIs are a serious concern for companies, which rely on such applications for business or for creating their public brand image.

Recent work on identifying XBIs [7, 3, 5] has proposed techniques which focus only on certain aspects of the web application's execution and are well suited for specific types of XBIs. For instance, the WebDiff tool [7] uses computer vision to detect XBIs, whereas CrossT [3] uses graph isomorphism along with text comparison to find XBIs. Hence these tools offer partial and imprecise solutions to the XBI detection problem. To address these limitations of existing techniques, we introduced the technique behind X-PERT, in [6], which integrates a rich set of comparison techniques and orchestrates them to apply each technique to the class of XBIs that it is best suited to detect. Our technique is an automated, precise, and comprehensive approach for XBI detection and is based on our findings from an extensive study of XBIs in real-world applications.

This demo paper discusses the architecture and implementation details of the X-PERT tool. It is organized as follows. Section 2 describes the different kinds of XBIs. Next, we summarize the technique behind X-PERT in Section 3. Section 4 presents details of the tool's implementation along with its usage scenario. The evaluation of X-PERT and related work are presented in Sections 5 and 6 respectively. Finally we conclude in Section 7.

## 2 CROSS-BROWSER INCONSISTENCIES

To establish a deeper understanding of XBIs, we performed a systematic study of 100 real-world web applications. Through this study, we were able to establish a classification of XBIs, which further helped us in defining our technique described in the next section. In particular, we

*This demo illustrates the implementation of a technique presented at ICSE'13 [6].


*ISSTA* '14, July 21-25, 2014, San Jose, CA, USA

found three main types of XBIs: structure, content, and behavior. These types are defined below.

***Structure XBIs:*** Such XBIs are observed in the structure, or layout, of individual web pages. The web page structure is essentially a particular arrangement of elements, which in case of structure XBIs, is disturbed in a particular browser. For instance, the misalignment of one or more web page elements on a given web page, in a particular browser, can constitute a structure XBI. We found that this was the most common category of XBIs, occurring in 57% of the subjects with XBIs.

***Content XBIs:*** This kind of XBI was observed in the content of individual components on a web page. Such differences can occur, where the visual appearance of a web page element, or the textual value of an element, are different across two browsers. We further classify these two cases as *visual-content* and *text-content* XBIs respectively. From our study, we found that these XBIs occurred on 30% and 22% of the sites with XBIs respectively.

***Behavior XBIs:*** These type of XBIs arise from the differences in the behavior of individual widgets on a web page. An example of such an XBI would be a button that performs a particular action within one browser and a totally different action, or no action at all, in another browser. Another example of such a case is the presence of an HTML link, which works in one browser but is broken in another one. In our study, such XBIs occurred on 9% of the web applications with XBIs.

In summary, Behavior XBIs affect the functionality of individual components, resulting in broken navigation between the different screens. On the other hand, Structure and Content XBIs are essentially differences in the arrangement or rendering of elements on a particular web page. In the next section, we describe how our technique detects each of these XBIs effectively.

# 3 TECHNIQUE OVERVIEW

Algorithm 1 presents an overview of our XBI detection technique. As shown in the algorithm, it takes as an input, the URL of the home page of the web application under test, *url*, and two browsers considered for the testing, $Br_1$ and $Br_2$. The technique outputs a list of XBIs, $\mathcal{X}$. Details of the algorithm steps are explained in our ICSE'13 paper [6]. We summarize the salient features of the technique here.

**Model Generation via Crawling:** The technique starts with crawling the web application, in an identical fashion, in each of the two browsers $Br_1$ and $Br_2$. In this process, it records the observed behavior as navigation models $M_1$ and $M_2$, respectively. The model is captured as a labeled transition system, which represents the top-level structure of the crawled web application. In the model, the states correspond to web application screens, and each transition is labeled with a widget action that leads to a screen navigation. In addition to this navigation model, we also capture the screen image and the DOM structure of the elements on each observed screen. In the algorithm, this step is implemented by the function *genCrawlModel* at line 3.

**Behavior XBI Detection:** The navigation models $M_1$ and $M_2$ are checked for equivalence to uncover differences in behavior. For this, the technique uses the graph isomorphism checking algorithm for rooted labeled directed graphs, proposed in [3]. This is implemented in the *diffStateGraphs*

**Algorithm 1:** X-PERT: Overall algorithm

**Input** : $url$: URL of target web application
$Br_1, Br_2$: Two browsers
**Output**: $\mathcal{X}$: List of XBIs

1 **begin**
2 $\quad \mathcal{X} \leftarrow \emptyset$
3 $\quad (M_1, M_2) \leftarrow genCrawlModel(url, Br_1, Br_2)$
`// Compare State Graphs`
4 $\quad (\mathcal{B}, PageMatchList) \leftarrow diffStateGraphs(M_1, M_2)$
5 $\quad addErrors(\mathcal{B}, \mathcal{X})$
6 $\quad$ **foreach** $(S_i^1, S_i^2) \in PageMatchList$ **do**
`// Compare matched web-page pair`
7 $\quad\quad DomMatchList_i \leftarrow matchDOMs(S_i^1, S_i^2)$
8 $\quad\quad \mathcal{L}_i^R \leftarrow diffRelativeLayouts(S_i^1, S_i^2, DomMatchList_i)$
9 $\quad\quad \mathcal{C}_i^T \leftarrow diffTextContent(S_i^1, S_i^2, DomMatchList_i)$
10 $\quad\quad \mathcal{C}_i^V \leftarrow diffVisualContent(S_i^1, S_i^2, DomMatchList_i)$
11 $\quad\quad addErrors(\mathcal{L}_i^R, \mathcal{C}_i^V, \mathcal{C}_i^T, \mathcal{X})$;
12 $\quad$ **return** $\mathcal{X}$

function on line 4, which produces a set of differences, $\mathcal{B}$, and a list *PageMatchList* of corresponding web-page pairs $S_i^1, S_i^2$ between $M_1$ and $M_2$. $\mathcal{B}$ contains a set of missing and/or mismatched transitions across pages, representing differences in dynamic behavior. Thus, $\mathcal{B}$ represents the behavior XBIs detected by the algorithm and is included in the output. *PageMatchList* contains the mapping between corresponding screens across browsers and is used to detect other kinds of XBIs (lines $6-13$).

**Matching screen elements:** To be able to find XBIs on two matched pages $S_i^1$ and $S_i^2$, X-PERT first computes a list of corresponding DOM element pairs in these pages ($DomMatchList_i$). This computation is performed by function *matchDOMs* (line 7) and is based on a *match index* metric for DOM element correspondence. The same metric has been used in earlier studies of XBIs [7, 5]. It is a value in the range $[0, 1]$, and is computed using a weighted combination of the element's DOM attributes, its XPath (*i.e.,* path in the DOM tree [1]) and a hash value computed from its descendants in the DOM tree. (See [5] for further details.)

**Structure XBI Detection:** A key contribution of the X-PERT technique was the notion of an alignment graph, which is an abstraction of the layout of a web page that captures the relative arrangement of all the elements on that page. X-PERT extracts alignment graphs from different renderings of a given web page on different browsers and compares them to detect structure XBIs. This is implemented by *diffRelativeLayouts* (line 8 in Algorithm 1), which compares pages $S_i^1$ and $S_i^2$ and extracts the set of relative-layout differences $\mathcal{L}_i^R$ that represent structure XBIs. More details about this technique can be found in [6].

**Text-content XBI Detection:** This step detects textual discrepancies in web page elements that contain text. To detect this class of XBIs, the text-value of an element is extracted from its DOM representation and compared with that of its corresponding element from $DomMatchList_i$. In the algorithm, *diffTextContent* (line 9) implements this checking and is computed over all the text bearing leaf nodes in the DOM tree. (For more details, see the *LDTD* feature for machine learning in [5].)

**Visual-content XBI Detection:** These XBIs represent differences in the visual appearance of individual page elements. For example, differences in the styling of text or background of an element across different browsers corre-

[1] `http://www.w3.org/TR/xpath/`

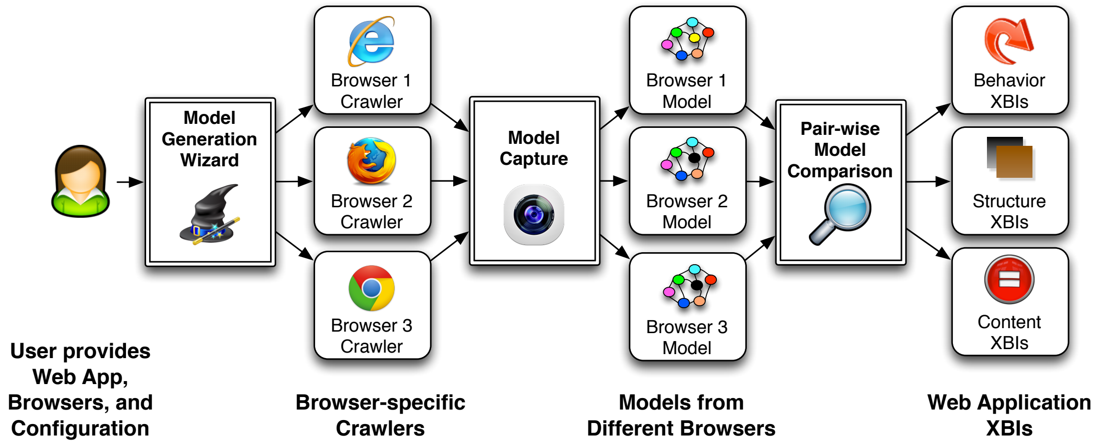


**Figure 1: High level overview of X-PERT.**

spond to visual-content XBIs. Such errors can only be observed in the image representation of the elements. Hence, the technique measures the $\chi^2$ distance between the color histograms of the element's screen image. This approach was also used in CrossCheck [5]. However, in X-PERT, we only apply this to the leaf DOM elements, where it is most effective at detecting visual-content XBIs. This operation is implemented by function *diffVisualContent* (line 10).

All the XBIs detected by the technique are added to the XBI list $\mathcal{X}$ (line 11) and reported to the developer.

# 4 TOOL DESCRIPTION

X-PERT can work with any web application, that runs on desktop browsers. Since it analyzes the client-side of such applications, it is agnostic to any server-side technology. The tool itself is written in Python and Java, and can run on a variety of desktop operating systems including Windows, Mac OS X, and Linux.

Figure 1 shows the high-level overview of X-PERT. It operates as follows. First the user invokes the web interface of the tool and interacts with its model generation wizard. This web interface of the tool is implemented in Python using the Flask framework (`http://flask.pocoo.org`) for the server-side, and Twitter bootstrap (`http://getbootstrap.com`) and jQuery (`http://jquery.com`) libraries on the client-side of the web application. Once the user submits the subject web application's URL and model capture parameters to the wizard, X-PERT then uses this information to generate different crawler instances, one for each browser. The generated models are then processed by the Model comparison module, which applies the technique to compare these models in a pair-wise fashion. This model comparison module is a key contribution of the X-PERT technique as it compares the different aspects of the web application's execution to uncover the three types of XBIs, which are then gathered, tabulated, and reported to the user.

The architecture of X-PERT, shown in Figure 2, consists of the model capture and comparison modules. Both these modules are mainly implemented in Java. Further details of the implementation are discussed below.

- The *Model capture* module uses the Crawljax tool [4], which internally uses the Selenium testing framework (`http://seleniumhq.org`) to explore the web application in the different web browsers. We extended Crawljax to save the model from its exploration along with the *screenshot* and *DOM structure* of each page. The DOM structure is obtained by querying the browser through its JavaScript interface and contains properties of each web page element's rendition. These properties include, for each element, its textual content, style information, its XPath location, its size, and its co-ordinates on screen.
- The *Differential Model Store / Loader* components are used to persist and load the models to/from the file system as XML files. This is implemented using the Object serialization support in Java, and is essential for the model capture and comparison modules to operate independently. For instance, model capture might be run to collect models from different machines, whereas model comparison would compare these models on a single machine.
- *Graph Isomorphism Checker:* This module performs an equivalence check between the graph based models from the two browsers to identify mismatched states and transitions across the two models. These models are implemented as Java objects and compared through a pair-wise traversal of the two graphs.
- The *DOM Tree Matcher* module matches corresponding elements on renderings of a web page across the different browsers by computing the *match index* metric. This metric considers the Levenshtein distance between the XPaths of the elements, and is computed using the corresponding implementation in the Apache StringUtils library.
- *Content Comparator:* The content comparator performs textual analysis of corresponding elements to detect text-content XBIs. For detecting image-content XBIs, it compares screen images of the corresponding elements on the

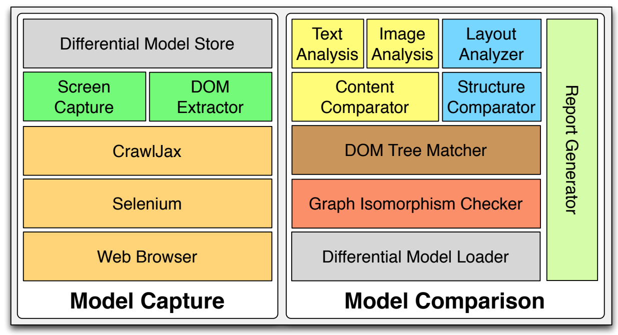


**Figure 2: Architecture of X-PERT.**

**Table 1: Results of X-PERT**

| Name | Behav. | | Struct. | | Content | | | | Total | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | Text | | Image | | | |
| | T | F | T | F | T | F | T | F | T | F |
| Organizer | 1 | 0 | 9 | 0 | 0 | 0 | 0 | 0 | 10 | 0 |
| GrantaBooks | 16 | 0 | 11 | 0 | 0 | 0 | 0 | 0 | 27 | 0 |
| DesignTrust | 2 | 0 | 5 | 3 | 0 | 0 | 0 | 0 | 7 | 3 |
| DivineLife | 7 | 0 | 3 | 6 | 1 | 0 | 0 | 0 | 11 | 6 |
| SaiBaba | 2 | 0 | 2 | 9 | 0 | 0 | 0 | 0 | 4 | 9 |
| Breakaway | 0 | 0 | 10 | 2 | 0 | 0 | 0 | 0 | 10 | 2 |
| Conference | 2 | 0 | 3 | 0 | 1 | 0 | 1 | 0 | 7 | 0 |
| Fisherman | 1 | 0 | 3 | 1 | 0 | 1 | 1 | 0 | 5 | 2 |
| Valleyforge | 0 | 0 | 2 | 2 | 0 | 0 | 1 | 0 | 3 | 2 |
| UniMelb | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 1 |
| Konqueror | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 6 | 0 | 6 |
| UBC | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| BMVBS | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| StarWars | 0 | 0 | 12 | 0 | 0 | 0 | 0 | 0 | 12 | 0 |
| TOTAL | 33 | 0 | 60 | 23 | 2 | 1 | 3 | 7 | 98 | 31 |

web page by leveraging the OpenCV toolkit [1]. Specifically, this image comparison measures the $\chi^2$ distance between their color histograms to detect image-content XBIs.

- *Structure Comparator:* The structure of the page extracted by the Model capture module is analyzed by the Layout analysis module to create alignment graphs, which represent the relative alignment of web page elements. Another graph isomorphism checker was implemented in Java to find differences in the alignment graphs of corresponding screens.
- *Report Generator:* This module generates an HTML report tabulating the set of detected XBIs. It is implemented using the Apache Velocity library (`http://velocity.apache.org`). The generated reports leverage jQuery along with the HTML5 Canvas for rendering and highlighting the XBIs on the screen images. Behavior XBIs are presented in the HTML report by highlighting them on the models converted to SVG format using the GraphViz tool (`http://www.graphviz.org/`).

# 5 EVALUATION

To assess the usefulness of X-PERT, we ran it on 14 subjects. These subjects are divided in three groups – the first six subjects were used in prior work, the next four were from our study, and, the final four subjects were obtained using an online random URL service (`http://www.uroulette.com/`).

Our experiments were performed using the latest stable versions of Internet Explorer (v9.0.9) and Mozilla Firefox (v14.0.1). Details of our empirical protocol can be found in [6]. The results of our investigation of X-PERT's effectiveness are shown in Table 1, which lists for each subject, the XBIs reported in the terms of true (T) and false (F) positives. As shown, X-PERT was effective in finding different kinds of XBIs from the subjects. A deeper investigation of the results [6] revealed that X-PERT's precision and recall is 76% and 95% respectively, against 18% and 83% for the state-of- the-art tool, CrossCheck [5].

# 6 Related Work

To the best of our knowledge, X-PERT is the first tool for comprehensive detection of XBIs. Previous research tools [7, 5, 3] only focus on certain types of issues and consequently have low overall precision and recall.

Developers typically access reference websites for browser-compatibility tables, such as `Quirksmode.org` and `CanIUse.com`, while developing web applications. Some web development tools such as Adobe Dreamweaver (`http://adobe.com/products/dreamweaver.html`) provide basic static analysis-based hints to detect certain issues. However, the issues targeted by reference websites and development tools are limited to only certain types of features, which are known to be missing in a particular browser. Other tools such as BrowserShots (`BrowserShots.org`) and Microsoft Expression Web SuperPreview (`http://microsoft.com`) provide previews of single pages in different browsers, while tools such as `CrossBrowserTesting.com` and `BrowserStack.com` allow browsing the web application in different emulated environments. However, the comparison of the observed behavior across browsers must still be performed manually.

# 7 CONCLUSION

Cross-browser Inconsistencies (XBIs) are a serious problem for web development companies. Current industrial practice relies on manual inspection to find these issues. Existing research tools only target particular aspects of the web applications and thereby report a significant number of false positives and negatives. To address these limitations, we presented X-PERT, which is an open source tool for comprehensive XBI detection. Our empirical evaluation shows the effectiveness of X-PERT over the state-of-the-art. This demonstration presents the details of the implementation of X-PERT and illustrates that it is fully automated, and easy to use through its web interface. In addition, it generates easy to comprehend and actionable reports for the developer to address XBIs effectively.

# 8 REFERENCES


[1] G. Bradski and A. Kaehler. *Learning OpenCV*. O'Reilly Media, September 2008.

[2] J. Lewis. Techniques for mobile and responsive cross-browser testing: An envato case study. `http://webuild.envato.com/blog/techniques-for-mobile-and-responsive-cross-browser-testing/`, July 2013.

[3] A. Mesbah and M. R. Prasad. Automated Cross-browser Compatibility Testing. In *Proceeding of the 33rd International Conference on Software Engineering (ICSE)*, pages 561–570. ACM, May 2011.

[4] 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*, 6(1):3:1–3:30, March 2012.

[5] S. Roy Choudhary, M. R. Prasad, and A. Orso. CrossCheck: Combining Crawling and Differencing to Better Detect Cross-browser Incompatibilities in Web Applications. In *Proceedings of the IEEE Fifth International Conference on Software Testing, Verification, and Validation (ICST)*, pages 171–180. IEEE, April 2012.

[6] S. Roy Choudhary, M. R. Prasad, and A. Orso. X-PERT: Accurate Identification of Cross-browser Issues in Web Applications. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 702–711. IEEE Press, 2013.

[7] S. Roy Choudhary, H. Versee, and A. Orso. WebDiff: Automated Identification of Cross-browser Issues in Web Applications. In *Proceeding of the 2010 IEEE International Conference on Software Maintenance (ICSM)*, pages 1–10. IEEE, September 2010.

[8] Stackoverflow. Posts for cross-browser issues. `http://data.stackexchange.com/stackoverflow/query/77488/posts-for-cross-browser-issues`, May 2014.