XFIX: An Automated Tool for the Repair of Layout Cross Browser Issues

Sonal Mahajan
University of Southern California, USA

Phil McMinn
University of Sheffield, UK

Abdulmajeed Alameer
University of Southern California, USA

William G. J. Halfond
University of Southern California, USA

ABSTRACT
Differences in the rendering of a website across different browsers can cause inconsistencies in its appearance and usability, resulting in Layout Cross Browser Issues (XBIs). Such XBIs can negatively impact the functionality of a website as well as users’ impressions of its trustworthiness and reliability. Existing techniques can only detect XBIs, and therefore require developers to manually perform the labor intensive task of repair. In this demo paper we introduce our tool, XFix, that automatically repairs layout XBIs in web applications. To the best of our knowledge, XFix is the first automated technique for generating XBI repairs.

CCS CONCEPTS
• Software and its engineering → Software testing and debugging; Search-based software engineering;

KEYWORDS
Cross-browser issues; automated search-based repair; web apps.

ACM Reference format:
DOI: 10.1145/3092703.3098223

1 INTRODUCTION
Companies invest a significant amount of effort to design and implement their websites. It is typical for companies to employ teams of graphic designers, web programmers, and testers to get a website’s “look and feel” correct across a plethora of browsers. This effort is important because inconsistencies in the appearance or usability of a website when rendered in different browsers, called Layout Cross Browser Issues (XBIs), can significantly impact users’ overall evaluation of a website; particularly, impressions of trustworthiness and reliability. Layout XBIs are discrepancies in the arrangement of HTML elements in a web page when rendered in different browsers, and can occur frequently due to differences in the implementation of HTML and CSS standards in different browsers’ layout engines. Layout XBIs are by far the most prevalent type of XBIs, observed in over 56% of the websites [10].

In general, debugging and repairing XBIs is challenging and has been a serious concern for web developers for a long time. For example, over 23,000 posts are reported for the search term “cross browser” on StackOverflow [12]. First, the sheer number of HTML elements and CSS properties that could be faulty, coupled with their complex interaction, makes it a difficult and labor intensive task. Second, repairing the XBIs by identifying correct fix values for CSS properties without introducing new XBIs is complex given the strong dependency between HTML and CSS, and their effect on a page’s layout. Third, developers need to repeat this task for every browser showing an inconsistency. XBI testing tools such as X-PERT [10] can only detect and identify faulty HTML elements; finding faulty CSS properties and fix values for a repair is still a manual effort.

To address these limitations, we proposed a novel automated technique [4] to repair layout XBIs in web applications. It uses search-based techniques to identify repair solutions. Our key insight is that the amount of layout deviation given by the position and size of HTML elements of a web page rendered in different browsers can be used as a fitness function to guide the exploration to likely solutions. When the amount of layout deviation converges to zero, the layout of a web page renders consistently in two different browsers, and a fault has likely been identified and repaired. In our evaluation, our technique was able to repair 86% of the XBIs detected by X-PERT and 99% of the XBIs observed by humans.

In this demo paper, we describe the implementation details of our repair technique [4]. We implemented our technique in a tool called XFix (available at https://github.com/sonalmahajan/xfix). To use XFix, users need to only provide a page under test and two browsers showing layout deviations (i.e., XBIs). XFix processes these inputs and modifies the page under test with a CSS repair patch that fixes the layout XBIs.

2 ENVISIONED USERS AND SCENARIOS
We envision the users of XFix as developers, testers, and support engineers of web applications seeking repair solutions for layout XBIs1. In this section, we discuss three usage scenarios in which XFix can assist its users in fixing XBIs automatically.

Repair Web Pages in Development. While implementing a website, web developers typically test it frequently on their “favorite” (reference) browser to ensure that it functions and renders correctly. After testing successfully on the reference browser, developers then typically test for XBIs by checking the website’s rendering compliance with other browsers. If XBIs are detected, developers spend time and effort in debugging and repairing the page. This manual effort can be saved by using XFix.

1Hereafter, we refer to layout XBIs as simply XBIs.
Repair Web Pages in Testing. After a web app has been implemented, it is typically sent to testing teams for verification and validation. Testers usually check for XBIs during User Interface (UI) testing, and if an XBI is detected, file a bug report and assign it to a developer for fixing. The developer then manually fixes the bug and reassigns it to testers for validation. This is a fairly time-consuming process that is potentially further complicated if the development and testing teams are not in the same geographic location. XFix can assist testers in this scenario by automatically repairing the detected XBIs, completely avoiding the loop of assigning a bug to a developer and re-validating the fix.

Repair Web Pages in Maintenance. Owners of legacy web applications (i.e., web applications in the maintenance phase of the software lifecycle) may want to maintain consistency of their web applications across newly launched browsers or newer versions of existing browsers. Production support engineers maintaining legacy projects are generally different to the original set of developers and testers, and may not have the required HTML expertise to repair XBIs. It may also be expensive to hire new developers solely for the purpose of fixing XBIs. In this case, XFix can effectively assist the production support engineers in repairing XBIs observed in existing web pages. The engineer can validate the repair and if satisfied, deploy the fixed page in production.

3 TOOL DESCRIPTION AND USAGE

The goal of XFix is to repair XBIs detected in a web page by finding potential fixes for them. XFix is implemented in Java and is a standalone tool that exposes a simple API to specify inputs and run the repair technique. To facilitate easy management of third-party dependencies, XFix is packaged as a Maven project. XFix can be run on any platform, such as Windows, Linux, and macOS. Since XFix analyzes the client-side code of the page under test, it is agnostic to the server-side technology used.

Figure 1 shows a high-level overview of XFix. A user of XFix is required to provide three inputs, a page under test (PUT), a reference browser (R), and a test browser (T). Users of XFix to provided R and T is a reasonable assumption in this domain, as they typically know the expected appearance of the PUT. For the PUT, the user needs to provide a URL pointing to the location of the HTML page on the file system where all the CSS, Javascript, and media necessary for rendering the PUT can be accessed. For providing R and T, XFix provides an option to select a browser from XFix’s supported set of browsers. XFix currently supports all the versions of the three most widely used browsers, Firefox, Chrome, and Internet Explorer (IE). More browsers can be easily added to XFix by including the browser’s standalone server implementation of the Selenium WebDriver’s wire protocol. Note that R and T need to be pre-installed on the user’s computer.

We now summarize the different stages of XFix’s technique in the following subsections along with their implementation details. Further algorithmic details of the technique are available in our ISSTA’17 paper [4]. Figure 1 shows the different stages of XFix in italics. Figure 2 is used as a running example to explain the different stages. The example shows screenshots of the menu bar of one of our evaluation subjects, IncredibleIndia, as rendered in IE (Figure 2a) and Firefox (Figure 2b). As can be seen, an XBI is present in the menu bar, where the text of the navigational links is unreadable in the Firefox browser (Figure 2b). This particular example was chosen because the repair is simple and easy to explain. However, most XBIs are much more difficult to resolve, with the repairs often spanning over multiple elements and CSS properties.

Stage 1: Initial XBI Detection. In this stage, XFix identifies the set of XBIs observed in the PUT rendered in T with respect to R. XFix uses Selenium WebDriver to open R and T and render the PUT in them, and then capture Document Object Model (DOM) information including XPath, location, and size of the elements. XFix then invokes X-PERT [10] by passing the two DOMs as input to obtain a set, X, of detected XBIs. X-PERT reports each XBI as the tuple \( \langle label, \langle e_1, e_2 \rangle \rangle \), where \( \langle e_1, e_2 \rangle \) is a pair of HTML elements in the PUT that is inconsistent in layout in \( T \) with respect to \( R \), and label describes the layout problem. As shown in Figure 3 for the running example of Figure 2, label \( \text{TOP ALIGNMENT} \) indicates that element \( e_1 \) is positioned on the top edge of element \( e_2 \) in \( R \), but not in \( T \). Another example of a label reported by X-PERT is \( \text{LEFT RIGHT} \) that indicates that \( e_1 \) is on the left of \( e_2 \) in \( R \), but not
in $T$. Note that we used the publicly available version of X-PERT (https://github.com/gatech/xpert) by making minor changes to fix bugs and add accessor methods for data structures. Due to space reasons, we describe the details of our changes to X-PERT on the XFix project page (https://github.com/sonalmahajan/xfix).

**Stage 2: Extract Root Causes.** In this stage, XFix extracts the root causes for each XBI detected in stage 1. A root cause is defined by the tuple $(e, p, v)$, where $e$ is an HTML element in the PUT, $p$ is a CSS property of $e$, and $v$ is the value of $p$. For each XBI, XFix first identifies a set of CSS properties relevant to the label descriptor. For doing this, we created a CSS-label map. For example, for the `TOP-ALIGNMENT` label, the CSS properties margin-top and top are relevant since they can alter the top alignment of an element with respect to one another. Note that this CSS-label map is an inherent part of XFix and does not require any further manual input from the user. Then for each CSS property, $p$, in the identified set, XFix adds two root causes, one for $e_1$ and one for $e_2$ (Figure 5).

**Stage 3: Search for Candidate Fixes.** In this stage, XFix performs a guided search over each of the root cause tuples populated in stage 2 to find individual candidate fixes. To do this, we implemented a search to be used in a single variable mode inspired by the variable search component of the Alternating Variable Method (AVM) technique [2]. The AVM is a local search strategy that first establishes a direction for the search, and then rapidly explores the space in that direction to find the optimal solution. We use the AVM technique as it accelerates the hill climbing search, thus helping achieve a quick convergence. XFix uses the AVM in a single variable mode by resetting the PUT to its original form before running the search on a root cause tuple to find a potential fix value, $v'$. For each root cause processed, XFix reports a candidate fix tuple $(e, p, v, v')$.

XFix applies each $v'$ to the PUT rendered in $T$ by using Selenium WebDriver to execute a Javascript function that dynamically modifies the DOM of a web page. XFix then extracts the updated DOM to compute the fitness score. The fitness function analyzes the relative layout deviation of $e$ in the PUT when rendered in $R$ and $T$ with respect to its local neighborhood. For quantifying the layout deviation, XFix considers three components: (1) difference in the location of $e$, (2) difference in the size of $e$, and (3) differences in the location of $e$’s neighbors. The fitness score is calculated as a weighted sum of the three components. The fitness function has a minimizing goal, i.e., a lower fitness score indicates a better candidate fix. We set the radius as 2 for defining the local neighborhood, and weights as 1, 2, and 0.5 for the three components, respectively.

Continuing with the running example, consider the processing of the root cause tuple $(e_1, \text{margin-top}, -20px)$. $e_1$ is located at $(10, 0)$ and its neighbor, $n_1$, at $(10, 100)$ in $R$ (Figure 4a). In $T$, $e_1$ is located at $(10, -20)$ and $n_1$ at $(10, 80)$, leading to a fitness score of 60 (Figure 4b). Trying $v' = 15\text{px}$ moves $e_1$ to $(10, -5)$ and $n_1$ to $(10, 95)$, resulting in a fitness score of 15 — indicating that the search is progressing in the correct direction (Figure 4c).

**Stage 4: Search for Best Combination of Candidate Fixes.** In this stage, XFix seeks to find a repair set for the PUT by finding an optimal combination of candidate fixes identified in stage 3. This additional search is necessary since not all candidate fixes may be required, as the identified individual fixes may have duplicating, multiplying, or competing effects. For example, candidate fix tuples, $(e_1, \text{margin-top}, 0\text{px}, 20\text{px})$ and $(e_1, \text{top}, 0\text{px}, 20\text{px})$ in our running example both result in a fitness score of 0 (perfect repair), however when applied together result in a multiplying effect introducing another XBI. For this stage, XFix uses a directed random search to identify the best combination. XFix selects a candidate fix with a probability $imp_{fix}/imp_{max}$. Here $imp_{fix}$ is the improvement observed in the fitness score when the fix was evaluated in stage 3 and $imp_{max}$ is the maximum improvement observed over all candidate fixes. XFix then dynamically applies each $v'$ in the selected fix tuples to the PUT, and uses the number of XBIs reported by X-PERT as the fitness score. XFix adds the fix tuples resulting in the lowest fitness score to the repair set. The search terminates when (1) the number of XBIs in the PUT equals zero, (2) the allotted resources (maximum number of fitness evaluations = 50) are exhausted, or (3) a saturation point is reached — i.e., $m = 10$ consecutive tries result in no fitness score improvement.

**Stage 5: Check Termination Criteria.** This stage determines whether XFix should terminate or continue searching for more repair alternatives. For this, XFix first applies the identified repair to the PUT to create the modified page PUT′, and runs X-PERT on PUT′ to identify new set of XBIs, $X'$. XFix terminates if the PUT is completely repaired, $X'$ shows no improvement compared to the previous iteration, or $X'$ reports more XBIs, making the current repair unviable. If none of the termination conditions are satisfied, $X'$ is assigned as the current set of XBIs, $X$, and PUT′ as the current PUT, and XFix loops back to stage 2 for the next iteration.

Upon termination, XFix generates a repair.css file (shown in Figure 6 for the running example) containing the repair patch and modifies the PUT file to include repair.css in the (head) section of the HTML of the page. Note that a new repair.css is created with a timestamp appended to the name of every run of XFix to effectively resolve XBIs across different test browsers for the PUT. XFix generates the repair.css as follows. First, XFix adds a browser specific qualifier corresponding to the test browser (e.g., -moz for Firefox) as shown in line 1 of Figure 6. Such qualifiers

1. \(<\text{HTML/BODY/}\text{DIV}[3]/\text{DIV}/\text{DIV}*, \text{margin-top, -20px}\>
2. \(<\text{HTML/BODY/}\text{DIV}[3]/\text{DIV}/\text{DIV}, \text{top, 0px}\.>
3. \(<\text{HTML/BODY/}\text{DIV}[3]/\text{DIV}, \text{margin-top, 0px}\.>
4. \(<\text{HTML/BODY/}\text{DIV}[3]/\text{DIV}, \text{top, 0px}\.>

Figure 5: Extracted set of root causes for the XBI in Figure 3

Figure 4: Fitness score computation in stage 3 — lower fitness score indicates a better candidate fix (i.e., fitness is to be minimized)
direct the layout engine to use the provided alternate values for the CSS property when it is rendered on a specific browser. For example, the repair patch shown in Figure 6 is only applied if the browser type is Firefox, and is ignored by the layout engines of other browsers. Developers widely employ this approach to repair XBIs. In our analysis [4] of the top 480 websites, 79% contained this type of browser-specific CSS code. Then, for each fix tuple \((e, p, v, v')\) in the repair set identified in stage 4, \(X\Fix\) converts the \(XPath\) of the element \(e\) to a CSS selector and adds it to the browser-specific qualifier block. For example, line 2 shows the CSS selector derived from the \(XPath\), `\://html/body/div[3]/div/div. \(X\Fix\)` then converts the fix value \(v'\), which is an absolute value (e.g., `margin-top: 20px`), to a relative fix value with respect to an element’s parent’s dimensions, such as `margin-top: 1.7%`. \(X\Fix\)` then adds this relative fix value for the CSS property \(p\) (line 3).

\begin{verbatim}
1. @-moz-document url-prefix() {
2.  html > body > div:nth-of-type(3) > div:nth-of-type(1) > div:nth-of-type(1) {
3.    margin-top: 1.7% !important; /* 20px */
4.  }
5. }
\end{verbatim}

Figure 6: repair.css generated for IncredibleIndia example of Figure 2

4 EVALUATION

In this section we summarize the results of our empirical evaluation obtained by running \(X\Fix\) on 15 real-world web pages [4]. \(X\Fix\) was able to resolve an average of 86% (median 93%) XBIs reported by X-PERT, and an average of 99% (median 100%) human observable XBIs. \(X\Fix\) required a median running time of 14 minutes. \(X\Fix\), on average, generates a repair patch of 9 CSS properties. This number is comparable to the browser-specific code size found in our analysis of 480 Alexa websites.

We also conducted a human study to measure the impact of \(X\Fix\) on the cross-browser consistency of the page. Based on user ratings we found that 78% reported an improved consistency of the page after the fix. This implies that \(X\Fix\) generated fixes that improved the consistency of the subjects across the different browsers.

5 RELATED WORK

Many search based techniques have been proposed in the field of software program repair that focus on Java and C programs [3, 15]. To our knowledge, \(X\Fix\) is the first technique that can repair presentation problems, such as XBIs, in web applications. Another technique [14] uses static and dynamic analysis to propagate a given client-side fix to the server-side source code. However, this technique cannot find the fix automatically.

Cross Browser Testing (XBT) techniques, such as X-PERT [10] and Browserbite [11], can be used to detect and localize XBIs. However, fixing the reported XBIs is still a manual effort.

Work in the field of web app presentation testing, such as WebSee [5–7] and FieryEye [8, 9] focuses on detecting and localizing presentation failures — a discrepancy in the actual and intended appearance of a web page. Another technique, GWALI [1], can detect layout failures in web pages after they are translated from one language to another. The ReDeCheck technique [13], uses a layout graph to detect potential layout faults in responsive web pages. These techniques are useful in detecting and localizing UI failures. However, fixing the detected failures is still a manual process.

6 CONCLUSION AND FUTURE WORK

In this paper, we presented \(X\Fix\), a fully automated tool for repairing layout XBIs in web applications. \(X\Fix\) uses a two-phased guided search approach to minimize the layout deviations observed in a page rendered in different browsers. The strong results of our evaluation indicate that \(X\Fix\) can be a useful and highly effective tool for automatically repairing layout XBIs in web pages.

In future work, we plan to extend \(X\Fix\) to support repair of cross-platform XBIs, such as across desktop and mobile environments. We also plan to produce repair patches for the CSS dynamically generated from other sources, such as Sass and server-side Javascript. Another direction we plan to work on is to automate the process of finding reference and test browsers from the set of \(X\Fix\) supported browsers. This can be possibly done by analyzing the XBIs reported by X-PERT for pairwise combinations of the browsers.

ACKNOWLEDGMENTS

This work was supported by NSF grant CCF-1528163.

REFERENCES