EvoWeb: Helping Users Discover the Evolution of the Web

Amanda Swearngin
Department of Computer Science & Engineering
University of Washington
Seattle, WA USA 98195
amaswea@cs.washington.edu

ABSTRACT
Billions of people use the internet daily to connect with friends, monitor their finances, conduct business, and keep up with social issues. Every time the web interfaces they use change, they can lose access to an integral part of their lives. To address this problem, we conducted interviews to determine the changes that users care about, finding that users focused on groups of change on a page. We developed and prototyped an algorithm to detect textual, visual, and structural groups of changes between two versions of a web page, allowing us to view a list of added, deleted, moved and changed user interface elements or groups. In a pilot study, our algorithm detected 54% of the changes that users perceived, and 76.6% of added and deleted elements between two page versions. Our study exposed the need for exploring smarter change detection that can semantically link textually similar elements on a page.

Author Keywords
Web applications; web page dynamics; change; re-finding

ACM Classification Keywords
H5.2: Information interfaces and presentation: User Interfaces – Graphical user interfaces.

INTRODUCTION
Every day, billions of people access websites that are constantly changing and evolving. The web is highly dynamic and evolves with surprising speed over time allowing people constant access to up to date information about the state of the world. However, because of this, it can be nearly impossible to find data, links, or navigational elements after the underlying page evolves.

Over eleven weeks, Fetterly et al. [6] found changes in 35% of a corpus of 150 million pages. Adar et al. [2] conducted a similar study and found that 65% of pages had changed over 5 weeks. These pages changed not only in the data they presented, but also in their interface design. Web developers remove, move, or re-label features without notice, leaving people searching for missing or re-labeled actions that they cannot recognize. Norman's "gulf of execution" [8] describes the gap between a person's intentions and the necessary actions in a system. Continuous changes constantly undermine a person's model of an interface, widening this gap until a person can no longer map their intentions to interface actions, leaving them confused and potentially unable to accomplish their goals. Figure 1 shows the changes made to Amazon.com in the span of one day, demonstrating the magnitude of change users must adapt to.

Figure 1. Amazon interface over two days. Taken from [9].

Given these motivations, we developed EvoWeb, a system that detects changed user interface elements in a web page. Through exploratory interviews, we found that users perceived changes in groups. We designed and prototyped an algorithm to compare two versions of a web page for structural, visual, and textual changes. It creates groupings of changes based on structural Document Object Model (DOM) location. In an evaluation, we found that our algorithm detected 54% of the changes that users perceived, and 76% of added and deleted groups of elements.

The contributions of this work are:

- An algorithm and prototype for detecting differences of added, deleted, moved, or visually changed elements or groups of elements between two versions of a web page.
- Evidence demonstrating that the algorithm we developed can mimic the groups of changes that users perceive and identify as change.

RELATED WORK
Existing tools can highlight differences in web page versions, but have mostly focused on content rather than interface changes. Arguably, one of the first, the AT&T Difference Engine [5] archives pages and displays

differences between consecutive versions. However, it considers only the DOM structure and textual content. Zoetrope [1] allows people to analyze a specific area of a web page (lense) for change over time, but focuses on data analysis, and not in helping users navigate interface changes. Zoetrope only collects screenshots of lenses over time, and does not perform a comparison for change detection.

Previous techniques for detecting change in web pages used tree alignment techniques to create mappings between nodes in the DOM tree [3,4,7,12]. Teevan et al. [11] presents DiffIE, a browser plugin that uses tree alignment to detect change, highlighting how pages have changed since the user last visited them. Although this system has many practical applications, it does not evaluate visual changes which can obscure content making changes unrecognizable to the user.

Bricolage [10] uses a structured prediction algorithm based on human-generated mappings to map two versions of a page. They consider visual, structural, and semantic constraints in creating the mappings; however, their technique targets completely different web pages, and not multiple versions of the same web page. Because we only want to compare two versions of the same webpage, tree alignment algorithms will work for our purposes.

EXPLORATORY INTERVIEWS

To guide the development of our change detection algorithm, we conducted five unstructured interviews to explore the factors to consider for a change detection and notification system. Participants were University of Washington CSE Ph.D. students. We gave users printouts of three categories of websites: news, banking, and number generation. We asked them to imagine they had a web page change notification system, and had them circle the areas of the interface that should be included. The key finding of our interviews was that people tended to group the areas of change, such as all the links in the website header or footer. Consequently, we designed our change detection algorithm to highlight the changes in groups, rather than individually.

Users expressed that the content in the headers and footers was important. For content in between, such as articles in a news website, results were more ambiguous. This is likely due to the transitory nature of the content. Page content (e.g., news articles) typically changes daily or weekly. Navigational elements like headers and footers typically change on version updates. The site's utility also had an impact. Users would not want notification for changes for unused user interface elements. Because the importance of a change may differ between users, we could filter out content or changes to unused elements over time.

CHANGE DETECTION ALGORITHM

Our change detection algorithm identifies changes at the Document Object Model (DOM) level, using the DOM content and the browser's runtime computed styles to expose the changes. The DOM defines the structure, style, and content of a web page. The DOM is interacted with through HTML, JavaScript, and CSS, which define the content, interactivity, and style of a web page, respectively.

Internally, a browser represents a DOM as a tree. Each user interface element or HTML tag is a node in the tree (Figure 2). Our algorithm operates on this tree, and the runtime *computed styles*, which is a list of the values of an element's CSS properties after applying the stylesheet and rendering the element. The algorithm compares two versions of a DOM tree, *V1* and *V2*, to find changed nodes or groups of nodes.

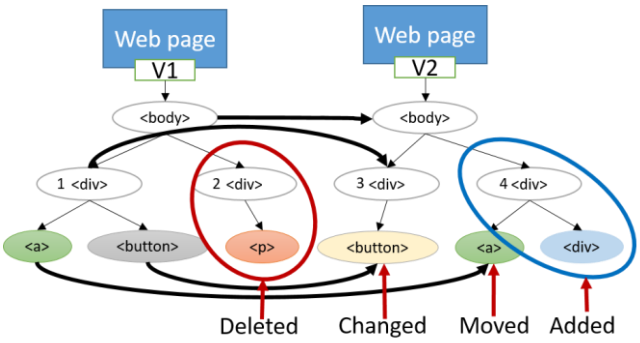

Figure 2. Change detection algorithm tree alignment.

Algorithm Goals

The goal of our change detection algorithm is to detect groups of added, deleted, moved, and changed user interface elements (shown in Figure 2). Each element corresponds to a node in the DOM tree. The groups we detect are as follows:

- ***Added***. The node did not exist in *V1*, and exists in *V2*.
- ***Deleted***. The node existed in *V1*, but does not exist in *V2*.
- ***Moved.*** The node has a different parent in *V1* and *V2*.
- ***Changed.*** The node has not moved but a computed style has changed (e.g., color, font).

Nodes in each category can be *leaf nodes*, which have no child nodes, or *non-leaf nodes*, which have child nodes.

Phase 1: Matching Leaf Nodes

In phase one of the algorithm, we attempt to match each leaf node in *V1* to a leaf node in *V2*. We determine if two leaf nodes match by their `tagName` (e.g., "div," "button", etc.), and `innerHTML`, the textual content of the node. If they match, we compare the *output affecting attributes.* Attributes are properties of the DOM node that are specified inline on the HTML tag (e.g., "disabled", "href"). An attribute is *output affecting* if its use causes a visual change to the DOM element when rendered. Examples of output affecting attributes for `<button>` are "disabled," and "hidden." For our purposes, we only identify changes visible to the user, so we do not compare any non-output affecting attributes.

After the algorithm compares these attributes, it identifies whether the leaf node is a match. The algorithm aligns leaf nodes as in Figure 2. At the end of the process, the algorithm has created a list of added, deleted, moved, and *matched* leaf nodes. Matched nodes have a matching leaf node in *V2*. We will compare them in phase three for visual differences.

Phase 2: Aligning Non-Leaf Nodes

In phase two, the algorithm matches ancestor nodes of leaf nodes to align the structure of the tree. This follows other tree alignment strategies [4,7,12]. Non-leaf nodes are matched by finding the node's children in *V1* and finding their corresponding locations in *V2*. The match is the parent node that has the most matching children in *V2*. For example, in Figure 2, node 1 matches node 3. In *V1*, node 1 has two child nodes, `<a>` and `<button>`. In *V2*, node 4 is the parent of `<a>`, and node 3 is the parent of `<button>`. Since there are an equal or greater number of children that exist in the original parent node location, node 3 is the match. The algorithm adds child nodes that are not beneath the original parent node to the `moved` list, and adds child nodes that did not exist in *V1* but exist in *V2* to the `added` list. The process continues until all non-leaf nodes are matched or placed in the `added` or `deleted` list.

Phase 3: Computing Visual Differences

In phase three, the algorithm iterates through the `matched` list and computes the visual differences. We obtain these from the elements computed styles. The algorithm compares each computed style value to its corresponding value in the matched node. We currently ignore a few specific computed styles: top, left, and position. This is because we wanted to categorize moved nodes separately, and define a move as a re-parenting in the tree hierarchy.

Phase 4: Creating Groups of Change

For the final step in the process, we create groups of change. For each node in the `added`, `deleted`, `moved`, and `changed` lists, if the parent of that node also exists in the list, we remove that node. At the end, only the top-level node in the hierarchy chain will end up in the list. In Figure 2, only node 4 will end up in the `added` list.

Results of Algorithm

Figure 3 shows the results of our algorithm on Ancestry.com. We prototyped our algorithm as a Google Chrome extension, and implemented it in JavaScript. It requires two versions of the page to be open in two tabs. We retrieved previous versions of sites from the Wayback Machine [9].

ALGORITHM EVALUATION

To evaluate our algorithm's effectiveness in identifying changes, we conducted a study to identify the changes that users perceived and compare them against the changes our algorithm identified to determine if there was a correlation. We had participants describe the groups of changes to help us key phrases associated with each group of change.

Participants

We collected data from nine participants (7 male, 2 female), recruited through an on-campus mailing list targeted toward graduate students. Two of the participants had worked as professional web developers. The rest of the participants had no professional experience or less than one year of web development experience.

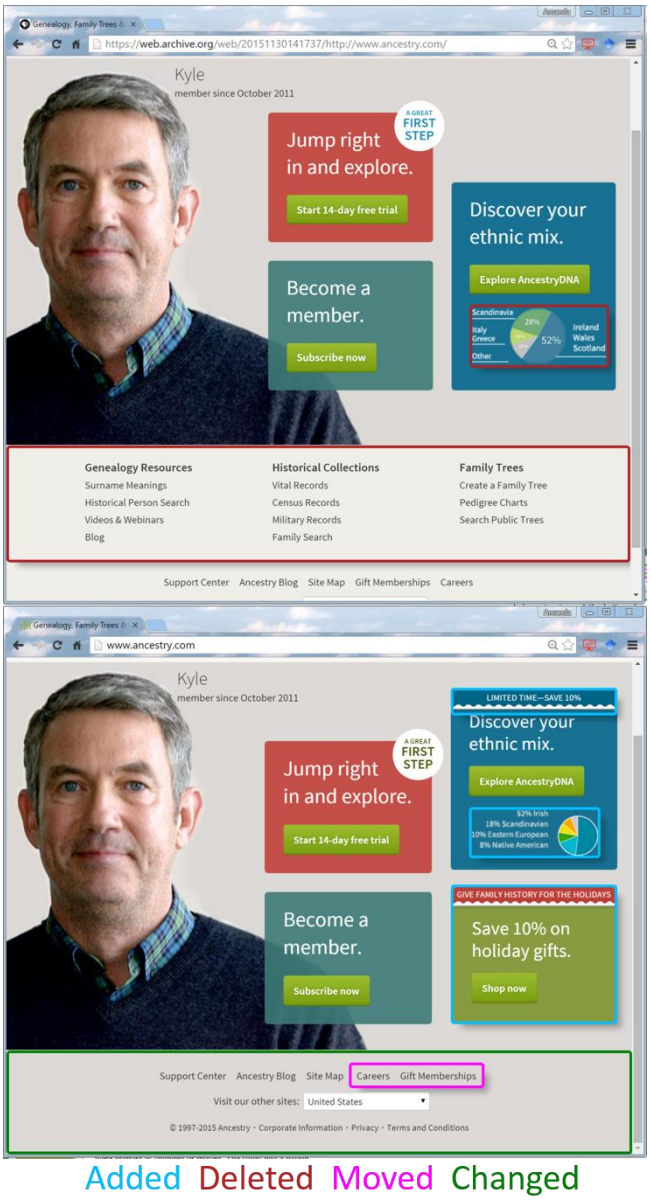

Figure 3. Results of algorithm on Ancestry.com website. Previous version of Ancestry retrieved from [9].

Apparatus

We gave each participant large, color printouts of before and after versions of web pages, marked with "before" and "after". Participants marked changes using colored markers. We selected before and after versions of three websites from the Wayback Machine [9], and Alexa.com top sites rankings.

Procedure

Each participant completed a two-part exercise. In part one, they marked the deleted user interface elements on the "before" version, and the added, moved, and changed elements on the "after" version. In part two, they gave a one-sentence description for each of the changes that they listed (not including the added, deleted, and moved changes). We do not report the results of part two as we were not able to collect enough data to observe any interesting patterns.

Design & Analysis

We construct the ground truth for comparison from the changes users identified. For each web page, we collected a list of the changes found by users and by EvoWeb. If the majority of users found a change, and EvoWeb found the change, we consider that a true positive. If the majority of users did not find the change, and EvoWeb found the change, we consider it a Type I error or false positive. The full error matrix that we evaluated EvoWeb against is in Table 1.

	Users found	**Users did not find**
EW found	True positive	False positive
EW did not find	False negative	True negative

Table 1. Full error matrix for evaluation.

RESULTS

For PayPal.com, EvoWeb found 48.1% of the same changes that users identified. But, it found seven additional changes that users did not find. Of the 14 differences that EvoWeb missed, eight were changes.

For Ancestry.com, EvoWeb found 50% of the same changes that the users identified, out of 6 total changes. There were also eight false positives. For Ancestry.com (Figure 3), the bottom links group with "Geneology Resources," and "Historical Collections," was marked as deleted. All participants also marked this as deleted. However, EvoWeb marked the "Start 14-day free trial," "Subscribe now," and "Explore AncestryDNA" buttons as deleted in *V1* and added in *V2*. All users marked these as a change in both versions.

For SimpleBots.co, EvoWeb found 66.6% of the changes that the users identified, out of 15 total changes. There were also no false positives found for SimpleBots, but there were still five false negatives. The slogan changed from "Nice Apps for Good People" to "Simple Apps for Complex People." The slogan remained in the same location. EvoWeb identified the slogan as deleted in *V1* and added in *V2*. All users marked the slogan as changed.

Overall, EvoWeb detected 87.5% of the deleted elements that users found across all three sites. EvoWeb also found 62.3% of the added elements that users found. EvoWeb also found a high number of false positives (7 adds, 8 deletes), corresponding to a set of changes users identified as the same element but changed. In total, users found thirteen changes and two moves not found by EvoWeb. Table 2 contains the full results of the changes found by EvoWeb and the users.

DISCUSSION

One issue we found is that EvoWeb does not detect changes that users can semantically link. For example, on PayPal.com *V1*, there was a slogan "Pay securely. Here, there, and everywhere." In the same location on *V2*, the slogan text changed to "Join the 173 million users already shopping with PayPal." Users tagged this difference as a change, while EvoWeb tagged it as a delete in *V1* and as an add in *V2*.

	PayPal Users		**Ancestry Users**		**SimpleBots Users**	
EW	**Y**	**N**	**Y**	**N**	**Y**	**N**
Y	**13**	**7**	**3**	**8**	**10**	**0**
N	**14**	**X**	**3**	**X**	**5**	**X**

Table 2. Results for each website. Y indicates the change was detected. N indicates the change was not detected.

Another issue that we found was that an elements internal tag type can change, causing our algorithm to miss a match. With Ancestry.com, we found that the internal tag type of some buttons had changed, causing them to be marked as deleted.

Users were also able to detect moves that EvoWeb could not detect, due to limitations of our algorithm. EvoWeb classifies a "move" as when an element moves structurally within the DOM tree. Users found a "move" for an element when its pixel-based location on the page changed significantly enough for them to detect it. Based on this finding, we might explore pixel-based location detection in the future.

FUTURE WORK

Currently EvoWeb highlights all detected changes. However, our studies indicated that users might not care about all changes. We plan to implement a customized filtering mechanism for each user; perhaps by collecting interaction traces to discover their most frequent actions. We would also like to explore alternate interfaces and presentations for presenting the changes. Currently, we simply highlight the detected changes. In the future, we will explore a more interactive tutorial-like interface that walks the users through their own set of relevant changes.

We would also like to address limitations to our matching algorithm. Our study indicated that our algorithm cannot semantically map together changed elements where the text changes because we use strict matching for text content. To address this, we plan to explore text similarity matching.

CONCLUSION

We presented EvoWeb, a system for detecting change in web. The system highlights groups of added, deleted, moved, and changed user interface elements. We found that our algorithm detects the majority of added and deleted groups that users identified, but does identify any changes that users were able to semantically link together, indicating a need for smarter mapping of textually similar elements. Building on these results, we will further improve the system, and work towards a solution to help users navigate the unceasing and endless changes made on the web every day.

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation (NSF) under grant CHS 1314399. We thank Daniel Epstein, Kyle Thayer, and Jacob Wobbrock for their feedback on early drafts. We also thank Andy Ko and James Fogarty for their helpful guidance on the research.

REFERENCES

1. Eytan Adar, Mira Dontcheva, James Fogarty, and Daniel S. Weld. 2008. Zoetrope: interacting with the ephemeral web. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology* (UIST '08). ACM, New York, NY, USA, 239-248. http://dx.doi.org/10.1145/1449715.1449756
2. Eytan Adar, Jaime Teevan, Susan T. Dumais, and Jonathan L. Elsas. 2009. The web changes everything: understanding the dynamics of web content. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining* (WSDM '09). ACM, New York, NY, USA, 282-291. http://dx.doi.org/10.1145/1498759.1498837
3. Kevin Borgolte, Christopher Kruegel, and Giovanni Vigna. 2014. Relevant change detection: a framework for the precise extraction of modified and novel web-based content as a filtering technique for analysis engines. In *Proceedings of the 23rd International Conference on World Wide Web* (WWW '14). ACM, New York, NY, USA, 595–598. http://doi.org/10.1145/2567948.2578039
4. Mira Dontcheva, Steven M. Drucker, David Salesin, and Michael F. Cohen. 2007. Changes in Webpage Structure Over Time. *Technical Report TR2007-04-02*, UW CSE.
5. Fred Douglis, Thomas Ball, Yih-Farn Chen, and Eleftherios Koutsofios. The AT&T Internet Difference Engine: Tracking and viewing changes on the web. *World Wide Web* 1, 1: 27–44. http://doi.org/10.1023/A:1019243126596
6. Dennis Fetterly, Mark Manasse, Marc Najork, and Janet Wiener. 2003. A large-scale study of the evolution of web pages. In *Proceedings of the 12th International Conference on World Wide Web* (WWW '03). ACM, New York, NY, USA, 669-678. http://dx.doi.org/10.1145/775152.775246.
7. Andrew Hogue and David Karger. 2005. Thresher: automating the unwrapping of semantic content from the World Wide Web. In *Proceedings of the 14th International Conference on World Wide Web* (WWW '05). ACM, New York, NY, USA, 86-95. http://doi.org/10.1145/1060745.1060762
8. Edwin L. Hutchins, James D. Hollan, and Donald A. Norman. 1985. Direct Manipulation Interfaces. In D.A. Norman, & Draper, S. W. (Eds.), *User-Centered System Design*, Hillsdale, NJ, USA, Lawrence Erlbaum Associates.
9. Internet Archive: Wayback Machine. Retrieved December 2, 2015 from https://archive.org/web/
10. Ranjitha Kumar, Jerry O. Talton, Salman Ahmad, and Scott R. Klemmer. 2011. Bricolage: example-based retargeting for web design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '11). ACM, New York, NY, USA, 2197-2206. http://doi.org/10.1145/1978942.1979262
11. Jaime Teevan, Susan T. Dumais, Daniel J. Liebling, and Richard L. Hughes. 2009. Changing how people view changes on the web. In *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology* (UIST '09). ACM, New York, NY, USA, 237-246. http://doi.org/10.1145/1622176.1622221
12. Yanhong Zhai and Bing Liu. 2005. Web data extraction based on partial tree alignment. In *Proceedings of the 14th International Conference on World Wide Web* (WWW '05). ACM, New York, NY, USA, 76-85. http://doi.org/10.1145/1060745.1060761