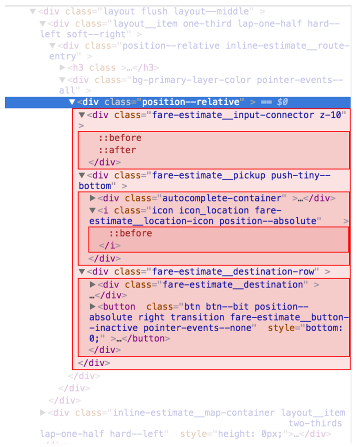
Ply: Visual Regression Pruning for Web Design Source Inspection



```
▼<div class="layout flush layout--middle" >
 ▼<div class="layout__item one-third lap-one-half hard--
 left soft--right" >
  ▼<div class="position--relative inline-estimate__route-
  entry" >
   ▶<h3 class >…</h3>
   ▼<div class="bg-primary-layer-color pointer-events--
   all" >
    ▼<div class="position--relative" > == $0
     ▼<div class="fare-estimate__input-connector z-10"
     >
        ::before
        ::after
      </div>
     ▼<div class="fare-estimate__pickup push-tiny--
     bottom" >
      ▶<div class="autocomplete-container" >…</div>
      ▼<i class="icon icon_location fare-
      estimate__location-icon position--absolute"    >
         ::before
       </i>
      </div>
     ▼<div class="fare-estimate__destination-row" >
      ▶<div class="fare-estimate__destination" >
      …</div>
      ▶<button  class="btn btn--bit position--
      absolute right transition fare-estimate__button-
      -inactive pointer-events--none"  style="bottom:
      0;" >…</button>
      </div>
     </div>
    </div>
   </div>
  </div>
 ▶<div class="inline-estimate__map-container layout__item
                                                two-thirds
 lap-one-half hard--left"  style="height: 0px;">…</div>
```

Figure 1: Ply's user interface extends the Chrome CDT, providing affordances for inspecting a component of the page.

Sarah Lim
Northwestern University
633 Clark St.
Evanston, IL 60208, USA
slim@u.northwestern.edu

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

Copyright held by the owner/author(s).
CHI'17 Extended Abstracts, May 06-11, 2017, Denver, CO, USA
ACM 978-1-4503-4656-6/17/05.
http://dx.doi.org/10.1145/3027063.3048427

Abstract

Despite the ease of inspecting HTML and CSS, web developers struggle to identify the code most responsible for particular stylistic effects, due to complex DOM structures and CSS property cascades. In this paper, we introduce Ply, a DOM inspection tool which augments the Chrome Developer Tools to help developers explore complex professional web designs. To compute source code relevance, we introduce a technique called visual regression pruning, which uses pixel-level screenshot comparison to help developers locate CSS responsible for a webpage's appearance. A user selects an element, and Ply computes the visual impact of each matched CSS property. If a property does not affect the webpage, Ply filters it from the inspector. In multiple iterations of needfinding studies, developers located relevant code more quickly using Ply. In a case study with the Airbnb homepage, Ply displays an average of 49% fewer CSS properties per element, compared to the Chrome Developer Tools as a control.

ACM Classification Keywords

H.5.2. [Information Interfaces and Presentation]: User Interfaces

Author Keywords

CSS; reverse engineering; web development; end-user programming

Introduction

Web developers often seek design inspiration from existing webpages, which offer developers the chance to learn from professional implementation decisions. However, developers struggle to locate and extract the specific lines of code responsible for their desired stylistic outcomes. Production CSS often includes frameworks such as Twitter Bootstrap,[1] or large common style guides containing thousands of globally-scoped properties. To find the code responsible for a webpage's appearance, developers must sift through complex HTML and CSS hierarchies.

Existing web inspection tools such as the Chrome Developer Tools[2] (CDT) offer element-based stylesheet inspection, but less experienced developers struggle to identify relevant properties from long lists of matched styles. Conversely, systems for CSS example reuse [2] provide greater instructional scaffolding, but do not expose the original source code. Snippets generated by these tools may procedurally mimic an element's appearance, but do not reflect style practices used by experienced web designers.

The conceptual contribution of this paper is the idea of *deriving relevant source code from the visual significance of a feature*. We illustrate this idea with Ply, a system for generating low-barrier learning materials for web design from complex professional webpages. Ply integrates with the CDT inspector to hide irrelevant CSS and provide affordances for inspecting DOM structures in isolation. To automatically identify irrelevant properties, we introduce a technique called *visual regression pruning*, which uses pixel-level screenshot comparison to compute the relevance of a range of code. Visual regression pruning starts by capturing a screenshot of the webpage, then disables the range of source code and captures another screenshot. If the two screenshots are identical, the code is pruned. Ply uses visual regression pruning to eliminate false leads from inspected CSS, helping developers identify relevant code more quickly.

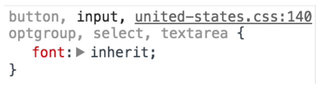
After reviewing related work, we present Ply's iterative design process and preliminary evaluation. During user testing, Ply helped student web developers locate and reproduce relevant CSS more quickly from complex professional webpages. We present a case study using Ply to reproduce a complex search bar from the Airbnb homepage. Ply displays an average of 49% fewer properties per element, compared to the CDT.

Related Work

Authentic learning [9] suggests that activities and materials should reflect real-world problems of personal interest to learners. In particular, Dorn et al. [3] show that web developers are driven to learn by project demands and the desire to remain up-to-date with standards. Despite the availability of professional websites as examples, however, developers struggle to gain authentic practice with existing inspection tools. Accordingly, our goal is to surface feature-relevant source code in a more meaningful way than simply displaying lists of properties for each element. Ply extends authentic example creation to web design, allowing developers to understand complex professional webpages of personal interest.

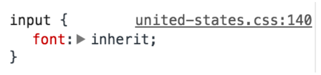
Recent work in *source discovery* explores the creation of authentic learning materials from the open web, allowing developers to learn from professionally written examples of their choosing. Telescope [6], Unravel [5], Scry [1], and FireCrystal [8] each contribute methods for guided

[1] http://getbootstrap.com/
[2] https://developers.google.com/web/tools/chrome-devtools/



```
button, input, united-states.css:140
optgroup, select, textarea {
  font:▶ inherit;
}
```

(a) CDT



```
input {        united-states.css:140
  font:▶ inherit;
}
```

(b) Ply

Figure 2: For matched CSS rules affecting multiple selectors, Ply only displays the selectors matching the current node.

discovery of source code for dynamic behaviors, but overlook the static analysis of CSS source for an element and its children. Specifically, Telescope allows users to see a small subset of JavaScript responsible for a dynamic effect, while Scry helps developers identify HTML and CSS differences during an interaction. Ply builds on this conceptual space with a novel technique for identifying relevant CSS for a static webpage feature.

Most relevant to our work, WebCrystal [2] simplifies CSS inspection by allowing developers to ask questions about an element's appearance, then generating a code snippet from its computed styles. Whereas WebCrystal hides the source style cascade to avoid overwhelming developers, Ply exposes the original source to promote authentic learning, with affordances to prevent information overload. In contrast to WebCrystal, Ply augments the professional Chrome Developer Tools to be more accessible to novices.

Visualization techniques from related systems help developers evaluate the visual output of their programs. Gliimpse [4] performs linear interpolation to animate a document from markup to output, but does not support CSS. *Visual regression testing* services such as Percy use screenshot comparison to detect breaking changes to a UI codebase. SeeSS [7] integrates real-time visual regression testing into an IDE. Ply introduces visual regression pruning to automatically determine relevance in visual source code. Our system targets developers interested in exploring professional webpages, where an existing codebase may not be available.

System Description

Ply augments the CDT element inspector by determining CSS property relevance, and simplifies the user interface with affordances for focusing on an individual DOM subtree.

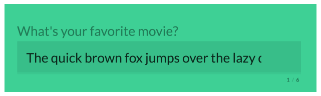
DOM inspection theoretically allows developers to learn by example, using any webpage of their choosing. In practice, current tools provide insufficient scaffolding for complex examples. Using the CDT element inspector, the developer must search through the entire cascade of matched styles, many of which are overridden or irrelevant to the current webpage state. To explore the component's internal structure in the DOM panel, she must keep track of its place amid hundreds of similar lines in the DOM hierarchy, even though she is only interested in the component's subtree.

Isolating a Component in the DOM Panel

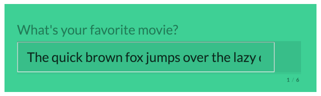
Ply's DOM panel highlights currently selected nodes, fading out ancestors and siblings. With full DOM hierarchy navigation available, a developer can visually isolate the subtree of interest. As the developer expands the node's children, Ply outlines each descendant subtree, using an opacity affordance to convey the current depth (Figure 1). For each node, Ply displays `id`, `class`, and `placeholder` attributes, since developers in needfinding reported using these attributes to distinguish between nodes in the DOM hierarchy. Other attributes, such as data identifiers injected by third-party frameworks, are hidden.

Inspecting CSS Styles

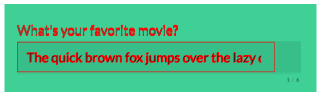
With potentially several dozen CSS properties applied to a single element, Ply displays the most relevant pruned subset of these properties, in descending order. Ply only displays properties whose removal changes the webpage visually. Developers can easily identify whether a property is directly applied or inherited using either of Ply's two formats. For properties defined in CSS rules with multiple selectors, Ply only displays the selectors matching the current node (Figure 2).


What's your favorite movie?
The quick brown fox jumps over the lazy

(a) Original image.


What's your favorite movie?
The quick brown fox jumps over the lazy

(b) After disabling `border: none;`


What's your favorite movie?
The quick brown fox jumps over the lazy

(c) Visual regression

Figure 3: Visual regression pruning uses image comparison to quantify the impact of removing a CSS property. In this example, we test the removal of `border: none;` on an input element.

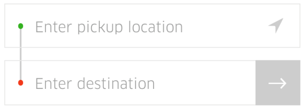


Figure 4: Developers reverse-engineered this form on the Uber homepage.

Implementation

Ply's UI affordances are implemented as modifications to the CDT front-end that communicate with a server to compute CSS property relevance using a technique called *visual regression pruning*.

Visual regression pruning determines whether a CSS property is irrelevant by disabling the property, capturing a screenshot of the resulting webpage, and comparing the screenshot to a baseline captured with all properties enabled (Figure 3). If there is no visual regression between screenshots, Ply removes the property from the CDT inspector.

Ply communicates with the browser using the Chrome Remote Debugging Protocol. When the developer selects an element for inspection, the server emits an event over the protocol. Ply responds by issuing a request for the element's matched styles, as well as a baseline screenshot of the webpage. For each matched property, Ply completes the following process:

1. Client issues an edit request to remove the property from the source stylesheet.
2. Protocol responds with a screenshot of the resulting webpage.
3. Client calculates the pixel-level difference between the new screenshot and the baseline, using the Mapbox Pixelmatch algorithm.[3] If the difference is zero, the property is flagged for pruning.
4. Client issues an edit request to restore the property in its source stylesheet, so that other properties can be tested independently.
5. If all properties have been checked, the client issues a request to hide the pruned properties from the CDT front-end.

[3]https://github.com/mapbox/pixelmatch/

Whereas prior systems rely on client-side computed styles [1, 2], Ply prunes and exposes the actual source styles for a webpage, allowing developers to learn from the author's implementation and architectural decisions. Computed styles often differ significantly from their source, do not reflect authentic development practices, and generalize poorly to alternate contexts. For instance, `width: 20%;` might be computed as `width: 233.33333px;` on render. The resulting code snippet obscures the author's decision to use a fluid-width layout, fails to demonstrate how properties cascade, and renders differently in a resized viewport.

Design Process and Insights

We conducted an initial need-finding study with six student web developers, followed by two iterations of prototype testing with two student web developers each. During needfinding, each developer spent 40 minutes attempting to replicate two components from complex professional webpages. Features included a full-screen responsive background image (Tumblr), a large multi-column footer (Slack), and a grid layout (Dribbble). In prototype testing, users spent 20 minutes attempting to replicate a form with custom-styled input fields (Uber) (Figure 4). Developers were compensated $15 each for the needfinding study, and $20 each for prototype testing.

Our initial needfinding study used the Chrome CDT as a state-of-the-art baseline. Showing the full DOM hierarchy and CSS cascade overwhelmed novice developers searching for relevant code: "When I first looked at [the CDT], I didn't know how to make this useful, and I still don't really know." Subsequent prototypes added simplifying affordances to the CDT element inspector. Our final prototype implemented visual regression pruning to reduce the number of properties shown in the CSS panel. We arrived at Ply's design based on two observations:

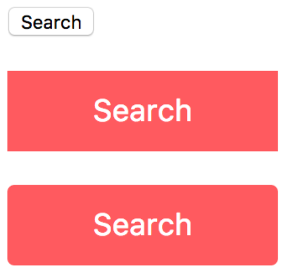


Figure 5: From top to bottom: an unstyled button with browser defaults, the results of Ply's pruning, and the original button extracted in its entirety from Airbnb. The pruned version bears strong similarities to the original, reduced from 43 to 20 individual CSS properties (a 53 percent decrease).

Element	Pruned	% Decrease
`BUTTON`	35	34.75%
`DIV`	6.47	54.01%
`INPUT`	36.33	35.03%
`LABEL`	10.4	57.80%
`SELECT`	40	35.71%
`SPAN`	12	38.1%

Table 1: Average CSS property reduction, by element name, from the Airbnb case study.

Ineffective Properties
Developers adopted a trial-and-error approach to CSS inspection, guessing which properties to copy into their editor, but many of these properties had no effect on the developer's output. The two developers who tested our first prototype copied 14 and 15 ineffective properties, eight of which were listed in the CDT as the most relevant to the selected element. These false leads caused universal frustration. Although the CDT allows developers to toggle individual properties on and off, this approach is time-consuming to carry out manually for an entire component. Ply reduces the guesswork associated with CSS inspection by only showing properties with a direct effect on the webpage. Of the two developers who tested our final prototype with pruning, neither copied any ineffective properties.

After successfully locating a key property using our final prototype, one developer reflected on the benefits of source inspection: "Usually when I'm using CSS I feel kind of stupid, like I'm just changing random numbers to guess how big it's gonna make it. So it's good just to see examples, like I can see the numbers they actually used so I don't have to guess around to try to get the same size that they got."

Keeping Track of DOM Position
We observed developers lose their position in the DOM inspector while switching back and forth between the webpage, inspector, and code editor. Within the DOM inspector, developers struggled to visually differentiate between the hundreds of similar-looking nodes: "There's just a lot here, I don't understand what any of these [attributes] mean." Since HTML is a hierarchical language, a node and its descendants often correspond to a component on a webpage. Ply uses color and outline affordances to help developers isolate and explore the subtree for a component. Developers found these improvements helpful compared to the CDT default: "If it didn't focus, I might just get lost where I was looking, and when I first open [the inspector] it's really overwhelming before it knows what I'm trying to focus on."

Case Study

To evaluate visual regression pruning, we conducted a case study on a professional website with an interesting CSS implementation. The Airbnb home page[4] contains a four-part search bar with block-level input labels, selection and input elements, and a large red "Submit" button.

Ply's CSS inspector view displays significantly fewer properties than the original source. We reconstruct the entire search bar using only 38 properties. As a point of comparison, the "Submit" button element contains 43 matched CSS properties in the original source code, but pruning results in a 53% decrease to 20 properties which suffice to construct a nearly identical button (Figure 5). Although the reconstructed version lacks rounded corners, Ply captures the element's core appearance with significantly less code. Across the 30 elements in the DOM subtree, Ply prunes an average of 13.5 properties per element, a 49.79% decrease. Table 1 shows the average number of properties pruned for each element type. Ply prunes the most properties from `SELECT` and `INPUT` elements (40 and 36.33, respectively), but `LABEL` elements had the largest percentage decrease (57.80%).

Discussion and Future Work

Ply demonstrates the feasibility of using visual regression pruning to support authentic exploration of complex webpage designs. In particular, Ply successfully hides ineffective CSS properties from the inspector, saving

[4]http://airbnb.com/

developers time by eliminating false leads. Whereas prior systems use computed styles to simplify the overwhelming CSS cascade, Ply's conceptual focus on authentic learning exposes the original source code, while filtering out irrelevant properties to minimize complexity.

A core limitation of visual regression pruning is the sensitivity of pixel-level image comparison. Examples of features which produce false positives in visual regression include: CSS keyframe animations, photographic backgrounds which are more prone to jittering the comparison algorithm, and non-deterministic interface elements, such as a calendar widget whose display changes depending on the date. Ply does not currently have any means for capturing state changes driven by JavaScript.

Our future work aims to create predictive models to generate the most likely relevance of CSS properties for a feature. By tracking the CSS implementation patterns of developers, we intend to understand whether the timing of CSS properties added to a DOM element correlates with the relevance of visual outcome from a property to its effect.

Acknowledgements

We thank Haoqi Zhang, Josh Hibschman, and the members of the Design, Technology, and Research program for their valuable guidance and feedback. This work was assisted by an Undergraduate Research Grant from Northwestern University's Office of the Provost.

References

[1] Brian Burg, Andrew J. Ko, and Michael D. Ernst. 2015. Explaining Visual changes in web interfaces. In *UIST 2015*. ACM, 259–268. http://doi.org/10.1145/2807442.2807473

[2] Kerry Shih-Ping Chang and Brad A. Myers. 2012. WebCrystal: Understanding and reusing examples in web authoring. In *CHI 2012*. http://doi.org/10.1145/2207676.2208740

[3] Brian Dorn and Mark Guzdial. 2010. Learning on the job: Characterizing the programming knowledge and learning strategies of web designers. In *CHI 2010*. http://doi.org/10.1145/1753326.1753430

[4] Pierre Dragicevic, Stephane Huot, and Fanny Chevalier. 2011. Gliimpse: Animating from markup code to rendered documents and vice versa. In *UIST 2011*. http://doi.org/10.1145/2047196.2047229

[5] Joshua Hibschman and Haoqi Zhang. 2015. Unravel: Rapid Web Application reverse engineering via interaction recording, source tracing, and library detection. In *UIST 2015*. http://doi.org/10.1145/2807442.2807468

[6] Joshua Hibschman and Haoqi Zhang. 2016. Telescope: Fine-tuned discovery of interactive web UI feature implementation. In *UIST 2016*. http://doi.org/10.1145/2984511.2984570

[7] Hsiang-Sheng Liang, Kuan-Hung Kuo, Po-Wei Lee, Yu-Chien Chan, Yu-Chin Lin, and Mike Y. Chen. 2013. SeeSS: Seeing what I broke – visualizing change impact of Cascading Style Sheets (CSS). In *UIST 2013*. http://doi.org/10.1145/2501988.2502006

[8] Stephen Oney and Brad Myers. 2009. FireCrystal: Understanding interactive behaviors in dynamic web pages. In *VL/HCC 2009*. http://doi.org/10.1109/VLHCC.2009.5295287

[9] David Williamson Shaffer and Mitchel Resnick. 1999. Thick" Authenticity: New Media And Authentic Learning. In *J. Interact. Learn. Res.* 10, 2: 195–215.