Poirot: A Web Inspector for Designers

Kesler Tanner
Stanford University
Stanford, USA
keslert@cs.stanford.edu

Naomi Johnson
University of Virginia
Charlottesville, USA
snj3k@virginia.edu

James A. Landay
Stanford University
Stanford, USA
landay@cs.stanford.edu

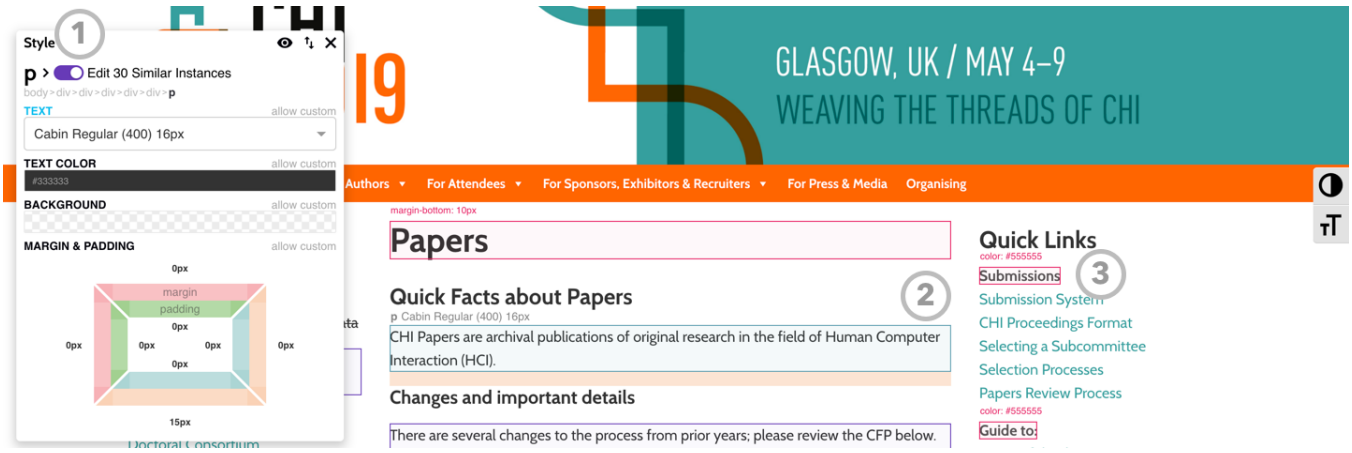

Figure 1: (1) The Poirot web inspection panel. (2) The element selected is highlighted in blue with a label providing additional details. Similar instances are highlighted in purple. (3) Previous changes are highlighted in red with labels showing changed properties and values.

ABSTRACT

To better understand the issues designers face as they interact with developers and use developer tools to create websites, we conducted a formative investigation consisting of interviews, a survey, and an analysis of professional design documents. Based on insights gained from these efforts, we developed Poirot, a web inspection tool for designers that enables them to make style edits to websites using a familiar graphical interface. We compared Poirot to Chrome DevTools in a lab study with 16 design professionals. We observed common problems designers experience when using Chrome DevTools and found that when using Poirot, designers were more successful in accomplishing typical design tasks (97% to 63%). In addition, we found that Poirot had a significantly lower perceived cognitive load and was overwhelmingly preferred by the designers in our study.

CHI 2019, May 4–9, 2019, Glasgow, Scotland UK

ACM ISBN 978-1-4503-5970-2/19/05.
https://doi.org/10.1145/3290605.3300758

CCS CONCEPTS

• **Human-centered computing** → **Graphical user interfaces**; **Web-based interaction**; Empirical studies in HCI.

KEYWORDS

Designer web tools; Inspector tools; Web development

ACM Reference Format:
Kesler Tanner, Naomi Johnson, and James A. Landay. 2019. Poirot: A Web Inspector for Designers. In *CHI Conference on Human Factors in Computing Systems Proceedings (CHI 2019), May 4–9, 2019, Glasgow, Scotland UK*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3290605.3300758

1 INTRODUCTION

More designers are finding themselves with a need to delve into the role of developer as end-user programmers, whether due to a lack of developer bandwidth in an organization, a desire to communicate visual details to a developer, or a need to optimize a design due to the difference in rendering between the original design tool and a web browser [12, 38]. For many designers, this foray from their world of visual design into a world of code and syntax is mandatory, but neither instinctive nor necessarily welcome [13].

Amiri has described the dissonance experienced by many designers working to understand development. He compares

programmers to linguists, trained professionally to understand the "syntax, semantics and pragmatics" of a language, while designers, in contrast, are more akin to tourists, interested in the language of coding only in so far as it is needed to "communicate with people[,] . . . explore the new environment[,] and . . . get by." [2]

There are tools to aid designers in navigating the developer space. Some of the better-established tools, however, denote by their very names that they have been lent from the developer's toolbox and were not created with the designer in mind. These are native browser web developer tools, such as Chrome DevTools, Firefox Firebug, and Safari Inspector.

Alongside these native web development tools, researchers have made new developer tools targeted at end-user programmers. WebCrystal [9] and CopyStyler [15] are both tools developed to assist novices. However, these tools focus on educating the novice in the comprehension and use of syntax, rather than on leveraging a designer's unique strengths.

Additional tools extend the capabilities of native web tools. While these systems could potentially be helpful to designers, they often address issues far beyond the capabilities of many designers. For example, while these tools are tackling issues like animation and interactivity [5, 23], designers are stumbling on much simpler problems like changing colors and font sizes. These problems might be trivial for a developer but are nonetheless hindrances to a designer.

In limited areas, the disconnect between development and design has been alleviated through web design tools such as Webflow, Adobe Dreamweaver, and other WYSIWYG editors. These tools allow a user to create a website using interfaces that are closer to traditional graphic design tools [36]. However, while these tools work well in small productions, such as a personal blog, they are not solutions typically used in professional environments, where developers prioritize maintaining greater control of their website code [38].

Therefore, new tools are needed that cater to the designer. For these new tools to be effective, a better understanding of the interaction between designer and developer in the modern professional ecosystem is critical. To help address these problems, we conducted interviews with several professional designers and developers to understand how they worked together to solve web development tasks. Based on the insights from these interviews, we created a survey and garnered responses from 43 professional UI/UX designers. Finally, we collected and analyzed professional design documents to confirm the findings of our survey.

Building on related work and our findings from our formative investigation, we designed and implemented Poirot (Figure 1), a web inspector tailored for designers that enables them to make style edits to websites using a familiar graphical interface. Poirot enables users to select an element on a website through direct manipulation, which displays a panel showing the element's current styles and allows modification of those styles. The tool assists the user in making consistent style choices by constraining the available options to those existing in the website's design system. Poirot remembers all updates to the website and can toggle back and forth between the original version and the modified version. When the user is satisfied with the results, Poirot can export these changes as a JSON file that can be imported by a developer for a live preview of the changes.

We evaluated the effectiveness of Poirot in a lab study with 16 professional UI/UX designers. We discovered that with no training, designers were able to successfully complete more tasks using Poirot than with Chrome DevTools (DevTools).

The contributions described in this paper are:

- A formative investigation of designers' web development experience that show that designers have close and frequent interaction with developers, often modify existing web interfaces, and play a leading role in maintaining the visual quality of websites.
- Poirot, a novel web inspection tool for updating the styles of existing websites, allowing designers to modify websites using a familiar graphical interface.
- A within-subjects evaluation of Poirot compared to a popular web inspection tool used for live edits of websites. This evaluation shows that when using Poirot users had significantly higher task completion rates, faster task completion times, a lower perceived cognitive load, and an expressed preference for Poirot interface to Chrome DevTools.

2 RELATED WORK

Prior research reveals that nonprogrammers approach development differently than programmers [37]. Indeed, web development, a growing form of end-user programming, has presented many challenges for teachers and curriculum makers [3, 11, 26, 38, 40]. Dorn studied graphic designers as end-user programmers, looking to address these difficulties [12]. His research provides insight into graphic designers' knowledge, understanding, and perspectives of programming [13, 14]. Park builds on this research, offering additional insight into the difficulties and errors novices experience when learning HTML and CSS [31, 33]. This collective research provides a foundation for understanding end-user programmers and has led to the creation of novel tools.

For example, OpenHTML is an interface that improves a novice's experience when writing HTML and CSS [32]. FireCrystal [30], Theseus [27], Clematis [1], and Telescope [24] help users identify code causing specific output. Scry [6] and Unravel [22] help users reverse-engineer the cause of behaviors in their code. Ply reduces complexity by hiding irrelevant code [28]. Tutorons provides inline explanations of

CSS and HTML code [20]. WebCrystal [10], Copystyler [15], C3W [17], and Marmite [25] enable users to combine code from different websites to assist novices to create mashups of ideas. Chickenfoot allows users to insert, remove, and replace functionality of live websites without access to the source code [4].

While these web development tools can be helpful to designers, they often require the user to be actively involved in the code. Designers have varying degrees of technical experience, and while some would identify as novice developers, not all designers are interested in using tools that were designed to improve the users' coding ability. Like existing tools, Poirot seeks to help designers make changes to websites. However, rather than encouraging or educating designers to learn more about development, it works to users' strengths by allowing them to modify websites through a graphical interface.

3 FORMATIVE INVESTIGATION

As Goodman et al noted, there is a "mismatch between HCI research and design practices" [18]; this has been the case for decades [19, 34] and has been discussed in various papers about interaction design [18, 35, 41]. To better understand how designers and developers cooperate in accomplishing their web development tasks, we interviewed four professional designers and developers working for multiple technology companies. Based on our insights from those interviews, we surveyed 43 UI/UX professionals working for a large technology corporation to better understand their design activities. Finally, we collected design documents and analyzed the types of changes requested by designers in those documents. These studies increased our understanding of tasks designers struggle to complete and highlighted new tool functionality that would be helpful to designers.

Interviews

We interviewed four professional designers and developers (two female, two male). Two identified as UI/UX designers, one identified as a software engineer, and the fourth was transitioning from the role of UI/UX designer to software engineer. All worked at large technology corporations in the United States. These interviews were unstructured and lasted up to an hour. One was conducted in-person while the others were done over Skype. Participants were not compensated for their time. During the interviews, we asked open-ended questions regarding the participant's current role. We also asked designers about situations in which they work with developers, and we asked developers about situations in which they work with designers.

From our interviews, we gained several insights into the designer/developer ecosystem in professional settings. First, we learned that designers and developers interact frequently, often on a daily basis, to accomplish design related tasks. Designers rely on developers to implement their designs and solve related problems. However, designers will choose to struggle through using inspection tools (e.g., DevTools) to solve issues on their own before interrupting a developer. If a designer is unable to solve the issue, she or he relies on help from a developer. An illustration of these close and frequent interactions is provided in the following example, which is a composite of what we heard in our interviews:

A designer starts creating a design mockup in a graphics tool. In these early stages, the designer converses with her developers to ensure her designs are feasible. When the designer completes the mockup, she adds additional markings (usually in red) to the design, which specify details about different design elements, like the number of pixels between items or the size and color of a heading. The designer then hands this "redline" document over to a developer, who transforms it into code for the website. During this phase, the developer and designer have several additional conversations to clear up any questions and add additional details.

After the design is implemented, it is published to the live site or staging server. At this point, the designer previews the design in her browser and confirms that the implementation was satisfactory. If there are issues, the designer attempts to fix them using DevTools. In situations where she is able to fix the problems herself, she documents the changes and communicates them to the developer. Depending on the size and number of issues, this communication happens verbally, by email, or through formal logging in a bug tracker, and is often accompanied by screenshots and/or the revised CSS or HTML updates.

In cases where the designer is unable to make the corrections herself using inspector tools, she then tries to find a time when the developer can work side-by-side with her. This enables the designer to watch the developer navigating DevTools while she explains what changes she wants to see. If the designer cannot get time with a developer, she logs the issue in a bug tracker without a documented solution.

Through our interviews, we also observed the efforts of designers and developers to maintain a site's integrity. Developers normally take the lead on code quality of a website, while designers oversee the visual quality of a website. Issues discovered during perusal of the site are treated with the same fix-and-document pattern described above. For example, one participant described a recent large and "grueling" undertaking in which he performed an audit of his company's entire site looking for instances where the company's design system [16] had not been followed. This audit consisted of browsing through the site looking for visible issues, as well as searching through the code base for the use of custom classes where these abuses often occurred.

Task	Selected
Add/decrease whitespace between elements	98%
Change the text size	98%
Change the text font	93%
Change the text color	88%
Change the background color	88%
Change the text copy	88%
Replace an image with another image	88%
Add or modify a drop shadow	81%

Table 1: Edits that designers from our survey realistically make when modifying a user interface.

Designer Survey

To confirm and deepen the findings from our interviews, we designed an online survey to gather feedback from a larger set of UI/UX designers. The survey was sent out on an internal Slack channel at a large technology company. Participants were entered into a raffle for a $50 Amazon gift card in exchange for their participation. Forty-three UI/UX professionals completed the survey. They had an average 6.6 (sd=5.7, min=2, max=25) years of experience as professional designers. Their average level of design expertise as defined by "How would you define your level of visual design expertise?" on a 7-point Likert scale from Novice to Expert was 5.5 (sd=1.1).

All participants reported having created UI design mockups as part of their job, and 36 said they regularly create UI design mockups. To the question, "What is your experience with creating redline design documents to give to a developer?", 21 reported that they regularly create redline documents to give developers, 15 reported that in the past they had created redline documents for developers, 5 reported that they were familiar with redline documents but had never created one, and 2 reported having never heard of redline documents.

On a 7-point Likert scale from "Strongly disagree" to "Strongly agree", participants averaged 6.2 (sd=1.1) for how often their work includes making improvements to existing user interfaces. To understand what kind of changes designers typically made, they were asked to select all changes they might realistically make when modifying a user interface. Table 1 shows the selection percentages.

Finally participants were asked, "In what aspects of your work do you interact with developers? What form does this interaction take?" These free responses confirmed our previous findings regarding developer/designer relations and interactions. The following participant's statement is reflective of answers provided:

> "Basically daily. They work on the designs I produce. I show them design stuff. Ideally we have a conversation about it (though not always unfortunately.) They work on it and come to me with questions (or just implement it how they like and wait for the QA later). I do QA on it."

Collection & Analysis of Design Documents

To verify that the types of changes designers said they made in the survey were actually what we saw in practice we collected over 50 redline design documents from a series of Google Image searches for keywords like "redline design document." Many of these documents came from technical write ups on UI designers' portfolio sites. We analyzed these documents for changes requested and notations used, categorizing the types of changes we found. While the style of redlining differed greatly, there was a commonality of changes. These changes supported the answers we received from the survey and interviews and most often included edits of color, content, shadows, typography, and whitespace.

Key Insights

Our investigation provided us with the following insights. These insights influenced our tool design and should be helpful to other future design efforts: First, discrepancies arise during the translation between the designers' mockups and the developers' HTML/CSS implementation, and designers often drive the efforts to correct these discrepancies. Second, designers struggle to remember CSS syntax and language. Third, designers struggle to know what elements are going to be affected by a change when modifying CSS. Fourth, designers tediously document their changes in an effort to minimize translation errors, and this documentation often requires the use of foreign terminology. Fifth, significant efforts are made to keep a website and design system in sync.

4 TOOL DESIGN

In this section we describe how we used the insights from our formal investigation to design and implement Poirot (Figure 1), a web inspector for designers.

In on our investigation, we found that errors frequently occurred during translation between the designers graphical mockup and the developers HTML/CSS. As a result, designers often attempted to use DevTools to correct these errors, but they struggled to effectively use this developer tool. These problems suggested that our tool should target the final medium but provide a user interface and interactions that were familiar to designers. For this reason, we built Poirot to work as a bridge between the designer and developer mediums, allowing designers to modify a live website using a familiar graphics interface while producing results

Figure 2: Poirot allows copying styles from one element and pasting the styles to another element. The resulting window lets designers toggle on and off properties to see a live preview of what styles are being transferred.

that are already in the medium of the browser. This permits designers to make the same pixel-precise changes they can make in graphics tools in the browser without worrying about their work getting lost during the translation to code since the translation is occurring in realtime while they work.

Poirot functions as a Chrome extension that can be injected into a page by clicking on the Poirot icon in the extensions bar. When Poirot is first loaded into the page, it takes inventory of all of the HTML elements, assigns each a unique ID, and categorizes them into text elements, image elements, and regular elements. These classifications are used to adapt the user interface and selection algorithms.

Poirot overrides default styling through CSS selector precedence by injecting a new stylesheet into the page and writing highly specific selectors (#poirot#poirot#poirot .poirot-256) with each CSS declaration using the !important tag. Unless a website is using highly discouraged !important inline declarations, Poirot selectors will have precedence.

Selecting Elements

Our formal investigation showed that when making CSS edits, designers struggled to know what elements were going to be affected. Poirot allows for a familiar graphical element selection experience. As the user hovers over elements, a semi-transparent gray box outlines the hovered element and a small label appears showing the element's HTML tag name. For text elements, the label also includes font family, font weight, and font size (Figure 1). When the user selects an element by clicking on it, the element's bounding box is highlighted in semi-transparent blue and the Poirot panel updates to show the properties of the element.

Poirot also shows other similar instances that will be edited *along* with the currently selected item. Poirot has different strategies for determining similar instances based on the type of element selected. If the element is a text element, Poirot finds similar instances throughout the page that have the same color, background color, font size, font family, font weight, and padding. Otherwise, Poirot looks for similar instances that are siblings of the selected element in the HTML structure and share background color, box shadow, margin, and padding, or elements throughout the page that have the same bounding box dimensions. These algorithms for determining similar instances worked well on the websites we tested, but they could be improved with even more targeted algorithms or machine learning.

These similar, auto-selected instances are outlined in a purple highlight. If the user desires to edit only the manually selected element, the auto-selection can be toggled off, and only the manually selected element will be edited.

Updating Styles & Content

Our formal investigation highlighted that designers struggle with remembering CSS syntax and language. Poirot reduces much of the complexity of updating styles and content by eliminating the need to remember CSS syntax and by handling logistical details for users.

Rather than each style being a property-value pair in a CSS rule, Poirot displays properties as UI widgets. As elements are selected, the Poirot panel updates the widgets it shows to display only those that are relevant. For example, if the element is an image element, the panel displays a "Change Image" button. If the element is a text element, it displays a typography dropdown selection (Figure 1).

For the browser to properly display fonts, the font needs to be installed locally on the machine, or the page needs to fetch the font file. Poirot handles this complexity by allowing the user to select the desired font from a list, then automatically makes the request for the font file. Currently only fonts served by Google Fonts are supported, but it could be extended to include other font services.

The same is true for image exploration. To display an image, the image needs to have a valid URL, either by being uploaded to a server or encoded as a Data URI. In Poirot, users can swap out an image for another by clicking on an image element and selecting the "Change Image" button. This opens a native file picker where the user can select the desired image. When the image is selected, Poirot converts the image to a Data URI and substitutes it for the current image.

Using standard copy/paste hotkeys, Poirot allows users to copy styles from one element and paste them onto another element. When the styles are pasted, a window shows the different properties that are being transferred (Figure 2). As toggles are turned on or off, the element receiving the style transfer updates in real time. When the user is satisfied with the transfer they can click to apply the styles. When a style

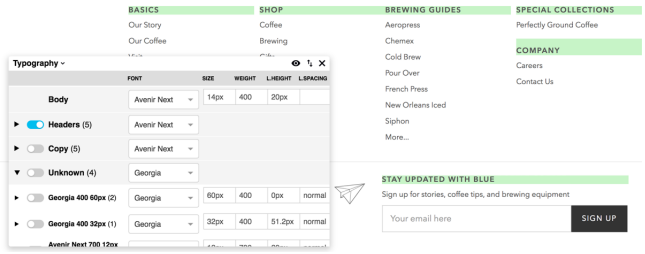

Figure 3: The Typography panel allows the designer to perform an audit to see where typography is being used and any unknown fonts that are not in the website's design system.

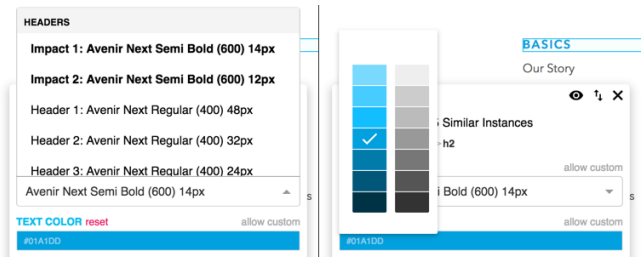

Figure 4: UI widgets for typography and color showing the built in design system choices.

transfer is complete, the user can choose a new element to act as a copy source and the same recipient element can be styled again. In this manner, the user could transfer the font size and font family from one element, the color from another element, and the margin and padding from a third element. Copying styles even works across elements from different websites, which is a useful ability since web designers often look at other people's websites, select the elements they like, and combine these in their own designs [21].

Maintaining Visual Consistency

Our formal investigation showed that significant effort is required to keep a website and design system in sync. To aid in this effort, Poirot has built-in support for design systems and currently supports colors, typography, shadows, and spacing [16]. Poirot's UI widgets help designers make consistent choices by only presenting values from the design system. If desired, users can break out of the design system by toggling "Allow custom" next to a UI widget.

In addition to helping the user make consistent choices when manually updating the page, the built in design system supports an interactive design system audit of a page. Figure 3 shows an audit of the typography used on a sample page. By turning on "Header", all header text elements are highlighted in green. Instead of turning on all headers, a specific type of header can be turned on, such as "Header 1". Under each type is a list of the elements using that type. Clicking on an element navigates to the element on the page.

The audit also shows *unknown* typography not defined by the design system. By going through these elements, a designer can decide whether to update the design system to include this unknown font or exchange the font for one defined in the design system.

Tracking Changes & Documentation

Our formal investigation revealed the tedious work of documenting requested website updates. Poirot eliminates this work by tracking all changes that are made to a page by a designer. When a property changes, the UI widget displays a "reset" button that allows the user to revert to the original value. Poirot also provides functionality to toggle on and off all modifications, allowing the user to quickly compare and contrast the updated page with the original page.

Page changes can be outlined by turning on "Redline changes". This highlights all changes in red and shows a label with the properties that changed and their new values (Figure 1). This notation mimics the notation styles we saw in our document analysis. When a designer is satisfied with their changes, they can export them as a JSON file to share with a developer or attach to a bug report. The developer can import the JSON file, turn on redlines, and visualize the updates. As she makes changes in the code that modify these values to match those made by the designer, Poirot will no longer highlight these elements. In this manner, developers can use the redline functionality as a checklist for making updates. When there are no more redlines, the developer has implemented all of the designer's changes.

5 USER STUDY

We evaluated Poirot in a within-subjects lab study in which participants used Poirot or DevTools to accomplish a series of design tasks. We selected DevTools as our comparator based on its popularity and feedback from our survey. For each task we measured task completion and task time.

Participants

Sixty-four participants volunteered for the study. All were recruited using convenience and snowball sampling from Facebook, Twitter, Slack, and Nextdoor. Recruitment messaging stated that we were seeking UI/UX designers. Participants filled out a pre-survey questionnaire containing questions about their design experience. Thirty-six participants said they had worked in the role of UI/UX designer, were comfortable or extremely comfortable using Sketch or Photoshop, and had created UI mockups or regularly created UI mockups. We emailed these 36 participants, and 16 participants signed up and participated in the in-person study. Nine were female. Participants reported an average 7.2 (sd=4.6, min=2, max=18) years of experience as UI/UX designers. Four participants

reported being extremely comfortable with DevTools, seven reported being comfortable, and seven had used DevTools but considered themselves beginners. Two participants reported being extremely comfortable with code editors, seven reported being comfortable with code editors, and seven reported using code editors but considered themselves beginners. All participants reported being familiar with design systems. The study lasted 60 minutes and participants received a $75 Amazon gift card.

Apparatus

The study was conducted on a MacBook Pro (Retina, 15-inch, Late 2013) laptop using the Google Chrome browser Ver. 67. Fifteen participants used the built-in trackpad and one used an optional external mouse that was provided. Scroll direction was adjusted to "natural" or "unnatural" to match the participant's normal scroll usage, and DevTools was positioned to "dock to right," "dock to bottom" or "undock into separate window" based on participant preference.

Tasks

The study consisted of nine design tasks. These tasks were selected based on our formative investigations, that showed that making these types of changes to existing user interfaces was common among UI/UX designers. As further confirmation, we asked study participants in a post survey how representative the study tasks were of actual tasks they might attempt to do in DevTools. On a 5-point Likert scale of "Not very representative" to "Very representative", they gave an average rating of 3.9 (sd=1.1). The tasks were as follows:

(1) Change the text color from gray800 to blue500.
(2) Change all four gray800 button backgrounds to blue500.
(3) Add a shallow drop shadow to both images.
(4) Change the text to "The Latest from Blue Bottle".
(5) Change the image to shop.jpg (file located in ~/Dropbox/Blue/).
(6) Change both kicker fonts from Impact 2 to Impact 1.
(7) Change the quote font family from Georgia to custom font Chelsea Market.
(8) Increase the vertical whitespace between the text and button from 10px to 40px.
(9) Decrease the vertical whitespace between links from 10px to 0px.

These design changes were all made on the same website. The website used for the study was a coffee shop website that we felt was representative of a modern website. It had a height of 3,766 pixels, and the browser window on the study laptop showed 803 pixels, resulting in 4.7 viewports.

To help participants understand where to make changes to complete each task, they were provided with a 22"x8.5" paper that showed a print version of the website as it should look after all the changes had been made. Next to each of the changes was a numbered red circle that indicated the corresponding numbered task.

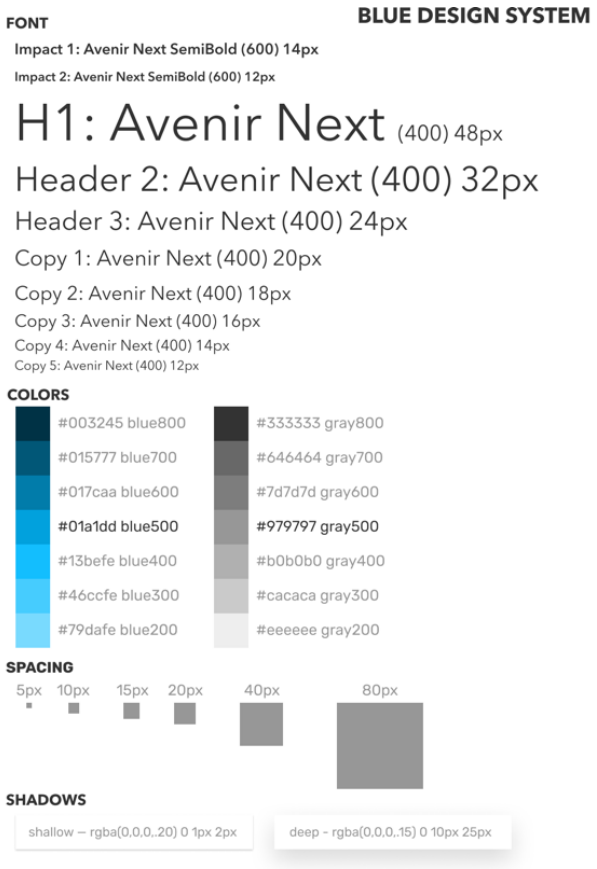

Figure 5: Design system used by participants.

In addition, participants were given a paper with a simple design system to use (Figure 5).

Our hypothesis was that Tool (Poirot/DevTools) would have significant effect on task completion and task time. Further, we wanted to explore task strategy as a moderator. We realized that tasks could be grouped into two categories based on the strategy that could be used to solve them: Find and Replace (F&R) and Custom. If a task was classified as F&R, it meant that a CSS rule existed targeting the exact set of elements that needed to be updated and the CSS rule contained a CSS declaration with the property that needed to be updated. If the declaration did not exist or the CSS rule targeting the set of elements did not exist, the task was categorized as Custom. Six of the nine tasks (2, 4, 6, 7, 8 and 9) were F&R tasks.

Procedure

Participants received a document describing the study procedure and introducing the design tasks, the design system, and the printed version of the website.

Participants were given three minutes to complete each task with up to an additional 30 seconds if they were in the middle of an attempted solution. Participants could skip a task at any time. They were also told they could use Google or any other online resource to complete the tasks. No training was provided for how to use either tool. The order in which participants used the tools was counterbalanced.

When a participant believed they had completed a task, they removed their hands from the keys and either raised their hand or verbally said "Done." The timer was then stopped and their work was checked. If it was correct, the time was recorded and the participant was asked to move to the next task. If it was incorrect, the participant was told what was incorrect, and the timer would continue. After attempting all nine tasks, the participant completed a standard NASA TLX questionnaire (excluding physical demand). Afterwards, the website was reset to its original state and the participant again attempted the nine tasks, this time with the alternate tool. After undertaking the nine tasks for the second time, the participant would again complete the NASA TLX questionnaire. Finally, participants completed a post-survey questionnaire asking about their experience with both tools. Following the survey, followup questions were asked to better understand the experience of using both tools, how participants used DevTools in their current work, and how they typically worked with developers.

6 RESULTS

We present our results as a function of Tool (Poirot vs. Devtools) and Strategy (F&R vs. Custom). The values on each bar in Figures 6-7 denote the number of data points.

Task Completion

To examine the effect of Tool on task completion, a logistic repeated-measures regression was conducted (Figure 6). Logistic regression was used since the outcome variable is dichotomous. There was a significant effect of Tool ($b=4.11$, $p<0.01$) indicating that the odds of completing a task was 20.36 times larger when using Poirot.

A logistic repeated-measures regression was conducted to examine the effect of Strategy on task completion (Figure 6). There was a significant effect of Strategy ($b=-2.63$, $p<0.001$) indicating that the odds of completing a task was 6.61 times larger on F&R tasks.

Furthermore, a logistic repeated-measure regression was conducted with F&R tasks to examine the impact of Tool on task completion. There was no significant effect of Tool

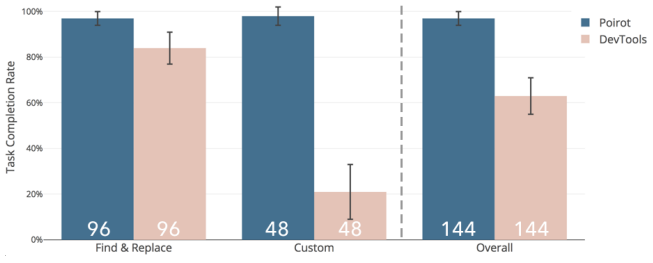

Figure 6: Average task completion rate per solving strategy and overall. Error bars represent 95% confidence intervals.

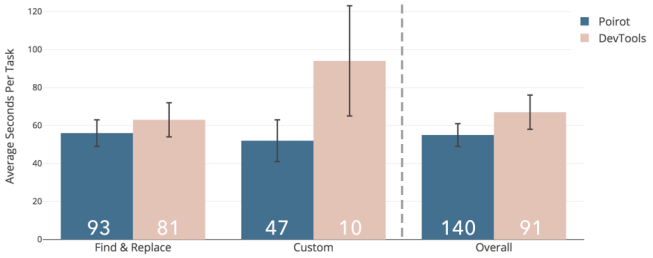

Figure 7: Average task completion time per solving strategy and overall excluding failed tasks. Error bars represent 95% confidence intervals.

($b=4.89$, $p>0.05$) indicating that for tasks that required F&R solutions, the odds of successfully completing a task was similar regardless of the tool used.

Another logistic repeated-measure regression was conducted with Custom tasks to examine the impact of Tool on task completion. The model failed to converge due to insufficient sample size, but the raw data provides suggestive evidence. The probability of completing a task was only 20.8% when participants used DevTools, compared to 97.9% when participants used Poirot.

Time

A Tool by Strategy repeated measures ANOVA was conducted on the time participants spent on each task.

There was a significant main effect of Tool ($F(1,15)=90.28$, $p<0.001$). On average, participants spent 61.5 seconds longer on the task when they used DevTools, compared to when they used Poirot. There was also a significant main effect of Strategy ($F(1, 15)=26.29$, $p<0.001$). On average, participants spent 35.0 seconds longer on Custom tasks compared to F&R tasks. In addition, there was a significant Tool-by-Strategy interaction ($F(1,15)=32.96$, $p<0.001$). The time saved by using Poirot (versus DevTools) was 78.4 seconds more on Custom tasks, compared to F&R tasks. Specifically, simple effect analysis showed that, on F&R tasks, participants spent 22.3 seconds longer when they used DevTools, compared to when they used Poirot ($F(1,15)=6.35$, $p<0.05$). On Custom tasks,

participants spent 100.7 seconds longer when they used DevTools, compared to when they used Poirot ($F(1,15)=102.8$, $p<0.001$). In other words, while there was advantage in using Poirot overall, this was much larger on Custom tasks.

A second Tool by Strategy repeated measures ANOVA was conducted on the time participants spent on solving each task, however, tasks that participants failed to complete were excluded from the analysis. Overall, the results were consistent with the analysis that included all tasks, but the effects appeared to be slightly weaker (Figure 7).

There was a significant main effect of Tool ($F(1,6.64)=9.64$, $p<0.05$). On average, participants spent 12.1 seconds longer to find a solution to the task when they used DevTools, compared to when they used Poirot. There was also a marginal main effect of Strategy ($F(1,5.61)=4.05$, $p=0.09$). In addition, there was a marginal Tool-by-Strategy interaction ($F(1,5.69)=5.50$, $p=0.06$). The time saved by using Poirot (versus DevTools) was 35.0 seconds more on Custom tasks, compared to F&R tasks. Specifically, simple effect analysis showed that, on F&R tasks, there was no significant difference in time participants spent to find a solution ($F(1, 14.66)=2.54$, $p=0.13$). On Custom tasks, participants spent significantly more time when they used DevTools to find a solution, compared to when they used Poirot ($F(1,5.34)=8.10$, $p=0.033$).

NASA TLX

Five of the six NASA TLX categories were used to determine a perceived cognitive load for each task: effort, frustration, mental demand, temporal demand, and performance. Each was rated on a 7-point Likert scale where 1=Very Low and 7=Very High. For the performance metric, the labels were adapted to 1=Perfect and 7=Failure. Perceived cognitive task load was calculated by averaging the individual scores from each category. Participants reported an average perceived cognitive load of 2.4 (sd=0.8) when using Poirot compared to 4.6 (sd=1.1) when using DevTools. A Wilcoxon-Pratt Signed Ranks test indicates that the difference in perceived cognitive load is significant ($Z=3.39$, $p<0.001$).

The following are all reported on a 5-point Likert scale. On a scale of "Not very often" to "Very often", participants rated how often they used Chrome Developer Tools in their design work a 2.8 (sd=1.2). On the same scale, they rated how often they would use Poirot in their design work a 3.9 (sd=1.1). A Wilcoxon-Pratt Signed Ranks test indicates that the difference between the two ratings is significant ($Z=-2.26$, $p<0.05$). On a scale from "Not very useful" to "Very useful", participants rated the usefulness of the built in design system in Poirot a 4.5 (sd=0.7). On the same scale, they rated the usefulness of the Poirot user interface a 4.1 (sd=0.8). On a scale from "Not very intuitive" to "Very intuitive", they rated the intuitiveness of the Poiret user interface a 4.1 (sd=0.8). Finally, on a scale of "Not very representative" to "Very representative", participants rated how representative the study tasks were of actual tasks they might attempt to do in Chrome Developer Tools a 3.9 (sd=1.1).

In the post-survey many participants expressed interest in using Poirot for their work:

> "Finally, a direct manipulation interface for designers to go in and tweak code with!" - P1

> "Overall, I think this tool is perfect. Just some small refines for the details, it would be very popular tool." - P13

Thirteen of the 16 users in their post survey left comments in their post-survey saying they found Poirot "simple", "intuitive", or "easy" to use:

> "[I]t's very easy to use, I can learn how to use this in one second. And this tool gives me a feeling that I'm using a design software like Sketch to modify the page in a familiar way." - P13

> "Easy to use, similar to other design tools i am familiar with" - P15

7 DISCUSSION

The probability of whether a participant would successfully complete a design task in the user study was highly dependent on two factors: the tool being used and the strategy that could be employed to solve it. When a participant was using Poirot, there was a very high probability of successful completion of any task. In fact, participants using Poirot were successful in almost all tasks, only failing 4 of 144 tasks across all participants. In contrast, when using DevTools, participants collectively failed 53 tasks, and the probability of successfully completing a task was highly dependent on whether or not the task could be solved using a Find and Replace strategy.

As an example, task #2, update the background color for all gray800 buttons to blue500, was successfully completed by all but one of the participants. When participants selected one of the gray800 buttons, most quickly discovered a CSS rule with the selector ".primary-btn" and declaration "background-color: #333;". To successfully solve this task, they changed #333 to #01a1dd, and all gray800 buttons updated to blue500.

In contrast, task #1, update the color of a single heading from gray800 to blue500, was only completed by five of the participants when using DevTools. When participants selected the heading, a CSS rule with a selector targeting just this element did not exist. Instead, there was simply a rule that targeted all headings throughout the page with the declaration "color: #333;". Most participants changed this to #01a1dd, but after making the change, they discovered that all headings throughout the page were now blue500.

Poirot reduced the confusion in tasks such as these by following good design principles including direct manipulation [39], a clear conceptual model, and visibility [29]. Other confusion it eliminated entirely by hiding unneeded details.

For example, Poirot made it easy for users to select an element and clear which elements were going to be updated. In contrast, users struggled to select elements in DevTools due to its indirect method for selecting elements. DevTools does support a direct manipulation method, but this is hard to find and was only used by one participant.

> "[Poirot] highlighted on screen what I was looking at and made it a lot more intuitive on what I was editing and making changes to. I think it's hard with Chrome/Safari developer tools to really know what you're looking at with 100% accuracy in a short amount of time unless you're really an experienced coder." - P6

Following the design principle of consistency, Poirot's interface always showed a consistent UI, regardless of the underlying CSS declarations. In contrast, DevTools required designers to scroll through a list of CSS rules searching for the one that had the declaration property they were seeking.

Participants also struggled with knowing the exact syntax required and made syntax errors even when using correct declarations. These issues have been well documented [31, 33], and our observations support their findings. Poirot hides this complexity of CSS syntax by removing it entirely.

In general, DevTools required four steps for completing a design task: selecting an element, finding where to modify the element, remembering CSS syntax, and evaluating the change to an element. Each of these steps presented complications to designers, either due to the difficulties of programming syntax or unfamiliar UI. Poirot assisted users through these complications by hiding complexity and the use of intuitive or familiar interfaces and interactions.

This paper adds understanding to the complicated roles of UI designers as web developers and the opportunity for improved communication with programmers [34]. Poirot is one attempt to address pain points we discovered during our formative investigation, but there remains more opportunity here. It also sheds light on designers' challenges with current web inspector tools. Though we specifically studied designers, many insights should generalize to novice developers of varying backgrounds. It also presents several UI solutions including copying and pasting styles across websites, auditing a website's use of a design system, and visibility and control over multiple or single element selection.

8 LIMITATIONS & FUTURE WORK

In its current state, Poirot cannot persist changes across site code updates that change the underlying HTML structure. Strategies for identifying consistent elements across HTML structure changes could be employed to create a system more robust to changes [7, 8]. Poirot handled scores of websites we tested it on including Google, Amazon, and Wikipedia, but it wasn't able to prevent Javascript click events on sites such as Twitter or YouTube, precluding most items from being selected on those pages.

In the future, Poirot could be extended in numerous ways, such as design system component library support, automatically extracting design systems from existing websites, better element positioning support, animation and interaction support, as well as a list of minor improvements suggested by participants during the study. Because Poirot reminded participants of their traditional graphics tools, they expected it to support all the features they are used to such as undo/redo, drag to reposition panels, and shift to multiselect.

Regardless of any limitations, the concepts upon which Poirot is built were shown to be helpful to designers and provide valuable insights into future tools. Designers were faster and more successful in making changes when using Poirot, and they perceived Poirot to be easier to use than DevTools. By creating tools that leverage the expertise of individuals rather than force them to adapt to programmers' tools, the diversity of individuals making valuable contributions to websites can increase. The same principles that helped designers improve the aesthetic quality of a website could be applied to making a website more accessible or inclusive. The benefit of these edits would be enormous if they could be crowdsourced and shared with others.

9 CONCLUSION

In the modern ecosystem, designers must often rely heavily on developers to visually update websites. To understand these pain points, we conducted interviews with professional designers and developers, surveyed professional UI/UX designers, and collected and analyzed professional design documents. This investigation provided insights that informed the design and implementation of Poirot, a web inspection tool for designers. We evaluated Poirot with professional UI/UX designers in a within-subjects lab study, and discovered that compared to Chrome DevTools, participants completed more tasks with Poirot and were enthusiastic about the prospect of such a tool to assist in their design work. We concluded with a discussion of the difficulties we observed designers experience when working with a popular web inspection tool. We hope to see these findings inform the creation of better web tools for designers and improve the communication between designers and developers.

ACKNOWLEDGMENTS

We would like to thank Hasso Plattner for funding this research and Spencer Reynolds for his insights.

REFERENCES

[1] Saba Alimadadi, Sheldon Sequeira, Ali Mesbah, and Karthik Pattabiraman. 2014. Understanding JavaScript Event-based Interactions. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*. ACM, New York, NY, USA, 367–377. https://doi.org/10.1145/2568225.2568268

[2] Faramarz Amiri. 2011. Programming as design: The role of programming in interactive media curriculum in art and design. *International Journal of Art & Design Education* 30, 2 (2011), 200–210.

[3] Alan F Blackwell. 2002. First steps in programming: A rationale for attention investment models. In *Proceedings of the IEEE Symposia on Human-Centric Computing Languages and Environments*. IEEE, 2.

[4] Michael Bolin, Matthew Webber, Philip Rha, Tom Wilson, and Robert C. Miller. 2005. Automation and Customization of Rendered Web Pages. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology (UIST '05)*. ACM, New York, NY, USA, 163–172. https://doi.org/10.1145/1095034.1095062

[5] Brian Burg, Andrew J Ko, and Michael D Ernst. 2015. Explaining visual changes in web interfaces. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. ACM, 259–268.

[6] Brian Burg, Andrew J. Ko, and Michael D. Ernst. 2015. Explaining Visual Changes in Web Interfaces. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology (UIST '15)*. ACM, New York, NY, USA, 259–268. https://doi.org/10.1145/2807442.2807473

[7] Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. 2003. Vips: a vision-based page segmentation algorithm. (2003).

[8] Deepayan Chakrabarti, Ravi Kumar, and Kunal Punera. 2008. A graph-theoretic approach to webpage segmentation. In *Proceedings of the 17th international conference on World Wide Web*. ACM, 377–386.

[9] Kerry Shih-Ping Chang and Brad A Myers. 2012. WebCrystal: understanding and reusing examples in web authoring. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 3205–3214.

[10] Kerry Shih-Ping Chang and Brad A. Myers. 2012. WebCrystal: Understanding and Reusing Examples in Web Authoring. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 3205–3214. https://doi.org/10.1145/2207676.2208740

[11] Parmit K Chilana, Celena Alcock, Shruti Dembla, Anson Ho, Ada Hurst, Brett Armstrong, and Philip J Guo. 2015. Perceptions of non-CS majors in intro programming: The rise of the conversational programmer. In *Visual Languages and Human-Centric Computing (VL/HCC), 2015 IEEE Symposium on*. IEEE, 251–259.

[12] Brian Dorn and Mark Guzdial. 2006. Graphic designers who program as informal computer science learners. In *Proceedings of the second international workshop on Computing education research*. ACM, 127–134.

[13] Brian Dorn and Mark Guzdial. 2010. Discovering computing: perspectives of web designers. In *Proceedings of the Sixth international workshop on Computing education research*. ACM, 23–30.

[14] Brian Dorn and Mark Guzdial. 2010. Learning on the job: characterizing the programming knowledge and learning strategies of web designers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 703–712.

[15] Michael J Fitzgerald et al. 2008. *CopyStyler: Web design by example*. Ph.D. Dissertation. Massachusetts Institute of Technology.

[16] Brad Frost. 2016. *Atomic Design*. Brad Frost.

[17] Jun Fujima, Aran Lunzer, Kasper Hornbæk, and Yuzuru Tanaka. 2004. Clip, Connect, Clone: Combining Application Elements to Build Custom Interfaces for Information Access. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology (UIST '04)*. ACM, New York, NY, USA, 175–184. https://doi.org/10.1145/1029632.1029664

[18] Elizabeth Goodman, Erik Stolterman, and Ron Wakkary. 2011. Understanding Interaction Design Practices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. ACM, New York, NY, USA, 1061–1070. https://doi.org/10.1145/1978942.1979100

[19] John D. Gould and Clayton Lewis. 1985. Designing for Usability: Key Principles and What Designers Think. *Commun. ACM* 28, 3 (March 1985), 300–311. https://doi.org/10.1145/3166.3170

[20] Andrew Head and Marti A. Hearst. [n. d.]. Tutorons: Generating Context-Relevant, On-Demand Explanations and Demonstrations of Online Code. ([n. d.]).

[21] Scarlett R Herring, Chia-Chen Chang, Jesse Krantzler, and Brian P Bailey. 2009. Getting inspired!: understanding how and why examples are used in creative design practice. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 87–96.

[22] Joshua Hibschman and Haoqi Zhang. 2015. Unravel: Rapid Web Application Reverse Engineering via Interaction Recording, Source Tracing, and Library Detection. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology (UIST '15)*. ACM, New York, NY, USA, 270–279. https://doi.org/10.1145/2807442.2807468

[23] Joshua Hibschman and Haoqi Zhang. 2016. Telescope: Fine-tuned discovery of interactive web UI feature implementation. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*. ACM, 233–245.

[24] Joshua Hibschman and Haoqi Zhang. 2016. Telescope: Fine-Tuned Discovery of Interactive Web UI Feature Implementation. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST '16)*. ACM, New York, NY, USA, 233–245. https://doi.org/10.1145/2984511.2984570

[25] Jason Hong and Jeffrey Wong. 2006. Marmite: end-user programming for the web *(CHI '06)*. ACM, 1541–1546.

[26] Andrew J Ko, Brad A Myers, and Htet Htet Aung. 2004. Six learning barriers in end-user programming systems. In *2004 IEEE Symposium on Visual Languages-Human Centric Computing*. IEEE, 199–206.

[27] Tom Lieber, Joel R. Brandt, and Rob C. Miller. 2014. Addressing Misconceptions About Code with Always-on Programming Visualizations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 2481–2490. https://doi.org/10.1145/2556288.2557409

[28] Sarah Lim. 2017. Ply: Visual Regression Pruning for Web Design Source Inspection. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '17)*. ACM, New York, NY, USA, 130–135. https://doi.org/10.1145/3027063.3048427

[29] Don Norman. 2013. *The design of everyday things: Revised and expanded edition*. Constellation.

[30] S. Oney and B. Myers. 2009. FireCrystal: Understanding interactive behaviors in dynamic web pages. In *2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 105–108. https://doi.org/10.1109/VLHCC.2009.5295287

[31] Thomas H Park, Brian Dorn, and Andrea Forte. 2015. An analysis of HTML and CSS syntax errors in a web development course. *ACM Transactions on Computing Education (TOCE)* 15, 1 (2015), 4.

[32] Thomas H. Park, Ankur Saxena, Swathi Jagannath, Susan Wiedenbeck, and Andrea Forte. 2013. OpenHTML: Designing a Transitional Web Editor for Novices. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems (CHI EA '13)*. ACM, New York, NY, USA, 1863–1868. https://doi.org/10.1145/2468356.2468690

[33] Thomas H. Park, Ankur Saxena, Swathi Jagannath, Susan Wiedenbeck, and Andrea Forte. 2013. Towards a Taxonomy of Errors in HTML and CSS. In *Proceedings of the Ninth Annual International ACM Conference*

on International Computing Education Research (ICER '13). ACM, New York, NY, USA, 75–82. https://doi.org/10.1145/2493394.2493405

[34] Cynthia Putnam, Aaron Reiner, Emily Ryou, Morgan Caputo, Jinghui Cheng, Mace Allen, and Ravali Singamaneni. 2016. Human-Centered Design in Practice: Roles, Definitions, and Communication. *Journal of Technical Writing and Communication* 46, 4 (2016), 446–470.

[35] Cynthia Putnam, Aaron Reiner, Emily Ryou, Morgan Caputo, Jinghui Cheng, Mace Allen, and Ravali Singamaneni. 2016. Human-Centered Design in Practice: Roles, Definitions, and Communication. *Journal of Technical Writing and Communication* 46 (06 2016). https://doi.org/10.1177/0047281616653491

[36] Jochen Rode, Jonathan Howarth, Manuel A Pérez-Quiñones, and Mary Beth Rosson. 2005. An end-user development perspective on state-of-the-art web development tools. (2005).

[37] Jochen Rode and Mary Beth Rosson. 2003. Programming at runtime: Requirements amp; paradigms for nonprogrammer web application development. In *IEEE Symposium on Human Centric Computing Languages and Environments, 2003. Proceedings. 2003.* 23–30. https://doi.org/10.1109/HCC.2003.1260198

[38] Mary Beth Rosson, Julie Ballin, and Jochen Rode. 2005. Who, what, and how: A survey of informal and professional web developers. In *Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on*. IEEE, 199–206.

[39] Ben Shneiderman. 1983. Direct manipulation: A step beyond programming languages. *Computer* 8 (1983), 57–69.

[40] Ye Diana Wang and Nima Zahadat. 2009. Teaching web development in the web 2.0 era. In *Proceedings of the 10th ACM conference on SIG-information technology education*. ACM, 80–86.

[41] Tracee Vetting Wolf, Jennifer A. Rode, Jeremy Sussman, and Wendy A. Kellogg. 2006. Dispelling "Design" As the Black Art of CHI. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '06)*. ACM, New York, NY, USA, 521–530. https://doi.org/10.1145/1124772.1124853