WebCrystal: Understanding and Reusing Examples in Web Authoring

Kerry Shih-Ping Chang
Human-Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15213
kerrychang@cs.cmu.edu

Brad A. Myers
Human-Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15213
bam@cs.cmu.edu

ABSTRACT
Examples have been widely used in the area of web design to help web authors create web pages. However, without actually understanding how an example is constructed, people often have trouble extracting the elements they want and incorporating them into their own design. This paper introduces WebCrystal, a web development tool that helps users understand how a web page is built. WebCrystal contributes novel interaction techniques that let the user quickly access HTML and CSS information by selecting questions regarding how a selected element is designed. It provides answers using a textual description and a customized code snippet that can be copied-and-pasted to recreate the desired properties. WebCrystal also supports combining the styles and structures from multiple elements into the generated code snippet, and provides visualizations on the web page itself to explain layout relationships. Our user study shows that WebCrystal helped both novice and experienced developers complete more tasks successfully using significantly less time.

Author Keywords
Web Authoring; Examples.

ACM Classification Keywords
H.5.2 [Information interfaces and presentation]: User Interfaces - Interaction styles; D.2.6 [Software Engineering]: Programming Environments - Graphical Environments; Design Tools and Techniques - User interfaces.

General Terms
Design, Human Factors.

INTRODUCTION
Increasingly, Internet users with minimal technical training are creating their own web pages. This involves not only authoring the content, but also designing the appearance and behaviors of the web pages. Good design is important to improve the readability and usability of the pages, and also to reflect the unique personality of their creator. Using templates or a "wizard" interface to build a web site are common ways to create a well-designed web page without writing any actual code [7]. However, these techniques cannot fully satisfy every user's needs since people have different requirements and aesthetic preferences for their web pages [13].

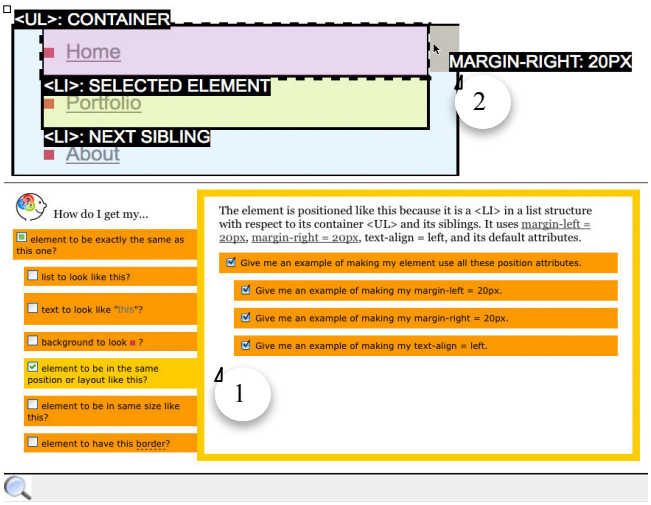

Figure 1: A WebCrystal user finds out how to lay out a list structure by inspecting an example web page, by (1) asking a position question of the selected list element in WebCrystal, and (2) inspecting the black area around the element on the web page.

Previous research has shown that one popular way for users to build a customized web page is through the use of examples, and that this results in pages with higher ratings [14]. It has also been reported that web designers in the real world often look at other people's websites, pick the pieces they like, and combine these pieces in their own designs [10]. In general, a popular way for developers to learn how to create code for any task is to look at examples [1]. One of the participants in our user study mentioned his own web design experience: "The best thing about web design is that all the code is open source… you can always go to the websites you like and see how they work." Potentially, every web page can serve as a design example for people who like some aspect of its design or want to learn how it achieved some effect. Assisting reuse of desired design elements from a web page could thus be very beneficial to many web designers.

Even after deciding on a design for a web page element, a person must know how to achieve that design. Thus, people must serve as both the *designers* for the style of a web page, as well as the *developers*, who write the code to implement the design. Reproducing a complex design from an example requires a person to have a decent knowledge about web development languages, even when using modern tools like Adobe Dreamweaver or Microsoft Expression Web. Furthermore, people often just want *part* of the design [10] and thus must identify which part of the code actually is responsible for the design aspect that is wanted. Consequently, to create a high-quality result, the ability to develop a web page to achieve the desired design can be as important as the ability to come up with a good design in the first place. Design examples are not useful if people cannot realize them in their own final work.

To help address this problem, we built *WebCrystal* (Figure 1), a tool that assists in the low-level programming tasks required to learn from and reuse examples as part of the web authoring process. "Examples" here can be anything built in HTML and CSS, the two most common languages used to create the static style of a web page. Currently, WebCrystal does not handle JavaScript, Flash, AJAX or other scripting languages. WebCrystal allows users to select any web element of a page that was created with HTML and CSS, and then choose from a set of "how" questions about how to recreate the different aspects of the selected element. The tool then answers the questions by (1) providing an automatically generated human-readable textual explanation using the element's HTML and CSS information, and (2) generating a customized code snippet for the users to recreate the design using the selected attributes.

WebCrystal makes the following contributions:

- A novel example-based web design tool for extracting and combining styling information from existing websites, and for helping designers understand how existing sites use HTML and CSS to create desired appearances. This includes how to achieve the positioning of elements.
- The novel use of automatically generated hierarchical questions and explanations about existing website styling information, in combination with element selection techniques to facilitate the extraction, combination, and understanding of existing website styling code. The explanations included generated code in a variety of user-selected formats (rather than code extracted from the source examples) that will reproduce the element.
- A between-subjects evaluation of the prototype system compared to standard tools for copying desired HTML and CSS from existing websites. This evaluation showed that WebCrystal users had significantly higher task completion rate and faster task completion time in reproducing web elements from a given web page, compared with people using Firebug [5], a state-of-the-art web development tool.

WebCrystal is inspired by previous systems Crystal [17] (which focused on desktop applications) and FireCrystal [19] (which focused on JavaScript). In WebCrystal, the goal is to make the construction of the HTML and CSS parts of web design "crystal clear" to the user, so that the user would be able to easily reuse them in their own pages. The tool was originally intended for novice and intermediate HTML and CSS developers, which we believe are the majority of the web authors today (and blog posts seem to indicate that even many "professional web designers" may not have particularly advanced coding skills—e.g., [8]). In fact, in our user study, we found that even advanced developers also benefited from and liked using WebCrystal.

RELATED WORK

WebCrystal was inspired by several previous systems that allow users to ask questions about a program's execution, as a way to provide more focused answers. Crystal [17] explains a word processor's behaviors by letting users ask about why elements look the way they do. Crystal not only provides a textual explanation, but also highlights the user operations that affect the element, so the user can more easily fix problems and control the operations. The Java Whyline [12] is a debugging tool that allows developers to ask "why" and "why not" questions about a Java program's output and leads developers backward to its causes. In WebCrystal, we adapted this idea of letting users ask questions in the different domain of web authoring.

WebCrystal allows its users to select the element they want from a rendered web page, and presents a code snippet that can recreate that element. Some prior systems have used a related approach in helping people make use of an example more efficiently by interacting with the output and displaying the corresponding source code. FireCrystal [19], for example, lets users record the interactive behavior of a web page and shows the JavaScript code that might be responsible for that behavior. Rehearse [2] is an extension for the Processing IDE that highlights each line of code in an example as it is executed to help programmers quickly identify which line of the code is relevant.

Many systems help with finding and presenting examples. Some systems allow users to specify layout information through sketches to retrieve desired designs [9], while others [14][15][20] support retrieving and browsing related designs from example galleries. WebCrystal helps *after* the examples have already been found, and assists users to incorporate desired examples into their own work. Some prior systems have focused on reusing examples without requiring users to look at or understand any code. CopyStyler [6] copies the style of text elements from one page to another. Adaptive Ideas [14] pre-annotates examples with metadata about properties and ranges of values, and then it can combine the user-selected attributes into a new page. Bricolage [13] maps some aspects of the style of one web page onto the content of another using an AI algorithm to rapidly generate a new web page. In contrast to these systems,

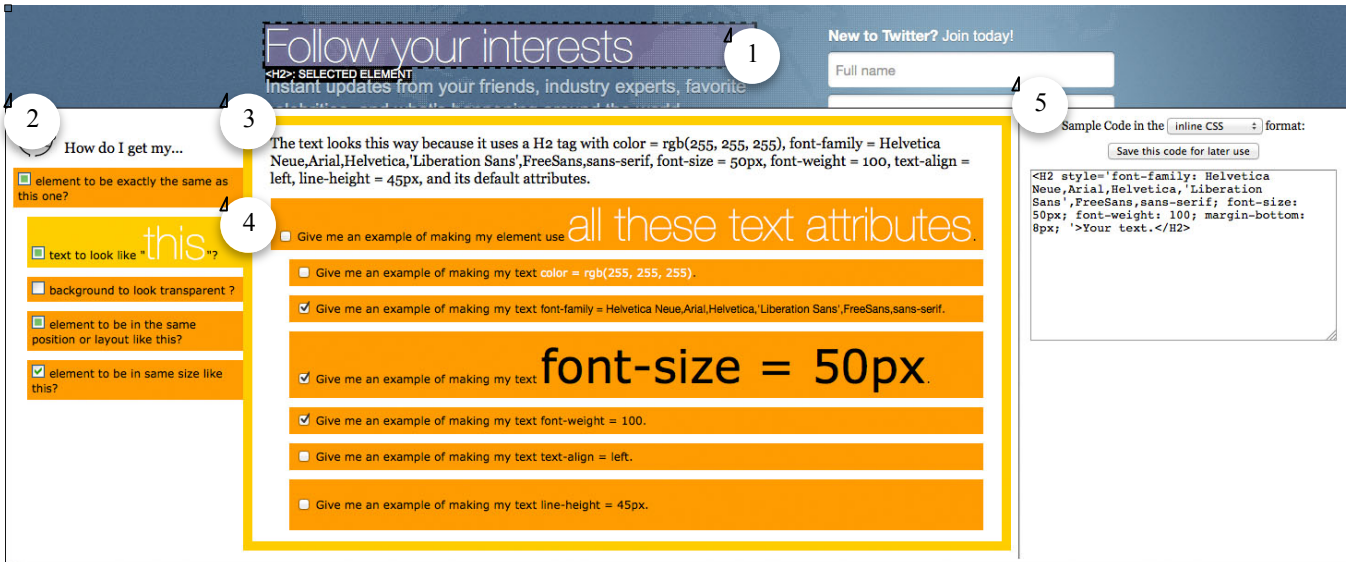

Figure 2: (1) is the web page under investigation, showing the purple highlight and the dotted box around the selected element. (2) is a list of questions the user can ask about recreating the selected element, with the selected question highlighted with a yellow background. (3) shows the textual response that answers the question selected in (2). Here, it explains the text properties. (4) is a sub-menu of commands that the user can pick to generate a code example using the selected properties. (5) shows the generated code.

WebCrystal gives users complete control of the results, since it does not use heuristic algorithms..It also exposes the underlying code to users, which might help them learn it.

THE WEBCRYSTAL USER INTERFACE

The main goal of WebCrystal is to enable the users to quickly reuse one or more aspects of an example in their own work. Ideally, WebCrystal would be integrated with a code editor, so the code extracted from the example could be directly inserted into the user's code. For now, in order to explore the appropriate user interface for discovering the appropriate code, WebCrystal instead runs as a plugin for the Firefox browser, and the user can copy-and-paste the code into a separate editor window. WebCrystal is a Firefox extension written in XUL [16] and JavaScript, using the jQuery [11] library. WebCrystal accesses the document object model (DOM) of the web page, and treats every DOM node in the web page as a web *element*. WebCrystal is activated by clicking on the magnifying glass icon button in the browser's status bar (Figure 1, bottom left). When activated, it tracks the current element under the mouse cursor as users move their mouse around the web page. The current element is highlighted using a semi-transparent box, with a small label showing the element's tag name (Figure 2 at 1). At the same time, a set of questions about how to recreate the various aspects of the underlying element are dynamically generated and displayed in the left of the WebCrystal window (Figure 2 at 2). The user can "freeze" a selection by left clicking on the desired element and then can start to inspect that element by browsing questions from the tool interface. A selection is "unfrozen" by left clicking anywhere on the web page.

Browsing the Elements Hierarchically

HTML is a hierarchical language. An element's position on the web page is related to its parents (containers) and siblings. For example, in Figure 1 the selected <li> is positioned relative to the position of its parent <ul>. Therefore, WebCrystal allows users to directly select any element using the mouse, but also allows users to move the selection around the hierarchical structure. Users can explore the hierarchy using the arrow keys to go up to the parents (containers) (e.g., from <li> to its <ul>), down to the children (<ul> to <li>), and left and right to the previous and next siblings (<li> to next <li>). In our user study, we found this feature also was useful for participants to select overlaid elements that have a similar size, by navigating up to parents and sideways to other siblings.

When WebCrystal is enabled, it disables the default function of mouse left button (navigating to hyperlinks) and arrow keys (scrolling the web page) so these events can be used to select and browse elements in WebCrystal. When the regular web page behavior is desired, WebCrystal can simply be disabled by clicking on its magnifying glass icon again to turn the tool off.

Asking Web Construction Questions

WebCrystal explains web construction by allowing users to select different "how" questions. All questions are in a similar form, such as "how do I get my … to be like this?" Previous studies in web design showed that when looking at an example, people only wanted *part* of the design and wanted to incorporate this specific part of the design in their own web page [10]. The key to successfully reusing an example is therefore to identify which lines of code are relevant to the user's needs, which is often difficult [2]. To address this requirement, WebCrystal generates questions on

both how to recreate an element as a whole, and how to recreate each aspect of the element. The first question in the list is always "how do I get my element to be exactly the same as this one?", which shows users how to recreate everything about the element. Next is a set of sub-questions that allow users to ask about recreating more specific aspects of the element. WebCrystal reads the HTML and CSS attribute values of the selected element and uses them to generate sub-questions in 11 categories (see Table 1). For questions about text, background, and border, WebCrystal also dynamically renders the style of the element's text, background and border, and shows it in the question description (Figure 2 at 4). Besides showing the properties immediately, we found this particularly important when the selected element's style is caused by *other* elements. For example, when DOM elements overlay on each other, the look of an element can be caused by the element behind it. Another example is that a vertical line can be made by using a background image inside a table cell or by turning on the left border of the next cell. By showing the feedback in the questions themselves, the user can select the correct element to ask the questions about. See Figure 5 for more examples.

After selecting a question, the middle pane will update and show the answer (Figure 2 at 3). The first part of the answer is a computer-generated human-readable textual explanation of what tag and attributes the element uses to generate the aspect(s) selected in the left pane. WebCrystal compares each attribute with its default value, and only shows the attributes that have a value different from their defaults, and thus would need to be specified by the user.

Generated Code

Under the textual explanation is a set of checkboxes with which users can select specific aspects of the property they are interested in (Figure 2 at 4). WebCrystal automatically generates an item for each CSS attribute that is relevant. When the user selects one or more of these, the system generates an example code snippet that will cause an element to have the same values for those attributes as the example. These choices are phrased as requests of the form: "Give me an example of making my element *attribute* = *value*." As a shortcut, the first checkbox says "Give me an example of making my element have all the same attributes" to get a result the same as the example. Attributes are also rendered in their style when possible to help users quickly identify what they want and understand the effect of the attributes.

The generated code snippet is shown in the code window on the right of the WebCrystal's interface (Figure 2 at 5). WebCrystal can provide code snippets in either inline-CSS or separate-CSS formats, chosen by the drop down menu, to let users generate the most appropriate kind of code for their situation (Figure 3). For cases where inline CSS is not allowed (such as for hover dynamic behaviors explained below), then the inline CSS option is removed.

Property for "How do I get my … to look like this?"	This question is displayed when:
Text	Tag is not <img>
Background	Every time
Position and Layout	Every time
Size	Every time
Border or Outline	CSS attribute "border-style" or "outline-style" is not "none"
Input element	Tag is "input", "select, "textarea" or "button"
Link	Tag is <a>
List	Tag is , , , <dd>, <dl>, or <dt>
Image	Tag is <img>
Table	Tag is <table>, <th>, <tr>, <td>, <thead>, <tbody>, or <tfoot>
Dynamic behavior	System detects there is a style change of the element before and after *mouseover* effect

Table 1: The 11 sub-question categories and their display conditions

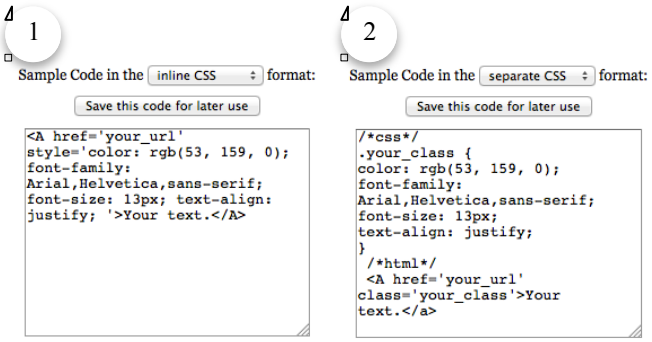

Figure 2: WebCrystal generates example code in both (1) inline CSS format and (2) separate CSS format.

The user can copy and paste the generated code into their editor to reproduce the effect. In addition, showing this code may help users learn the appropriate coding techniques. In a study of online code example usage [1], researchers found that using example code not only helped programmers finish their tasks more quickly, but also helped them learn new knowledge and clarify existing knowledge.

Explaining Position and Layout

WebCrystal provides additional questions and visualizations to explain the position and layout of elements by highlighting both the selected element's parent (container) and

siblings on the web page (Figure 4 at 1), and allowing users to inspect the blank areas around the selected element that are generated by CSS layout attributes such as margin, padding, top, left, bottom, and right. Users can inspect a blank area by hovering the mouse on it, and WebCrystal will display a label with the CSS attribute that caused this blank space, along with its value (Figure 4 at 3). Alternatively, if the user hovers the mouse over the CSS attribute in the textual explanation on WebCrystal's interface, then the corresponding blank space will be highlighted (Figure 4 at 4).

Explaining Dynamic Behaviors

A key feature of web pages is that they are interactive, not just static drawings. CSS adds the ability to support hover behaviors that show different styles depending on the mouse location. Therefore, WebCrystal adds additional questions so the user can ask for an explanation and code for these behaviors.

For hover behaviors that change the style of an element, the user can demonstrate to WebCrystal the styles by first selecting the element and then moving the cursor on and off of the element, to cause the style change. During the interaction, WebCrystal detects that the element changes styles. In this case, WebCrystal will show a message in the textual area that tells the user that this is a dynamic element, and which style of the element is being used right now. The message also includes a toggle button with which the user can see the other style for this element; that is, alternating between showing a description of the hovered and not-hovered styles for a link. The questions also toggle so the user can find out the details of how each of the styles is achieved. WebCrystal also adds a question so the user can ask "how do I get my element to have this dynamic effect", and will then provide the HTML and CSS skeleton code for the appropriate hover effect as the answer (Figure 5 at 2).

Figure 3: When answering position questions, WebCrystal (1) highlights the container and the siblings of a selected element, and (3) allows user to inspect blank areas around the element on the web page or (4) inside the textual explanation using the mouse.

Storing and Combining Multiple Elements

Many web design examples consist more than one DOM element. For example, the 3-grid menu structure in Figure 6, or the text and image layout in Figure 4, both use multiple elements to achieve those designs. The positioning of these elements results from their CSS layout attributes and their nesting in the HTML hierarchy, such as being siblings or parent and children. WebCrystal enables users to investigate this kind of layout design of multiple elements, and

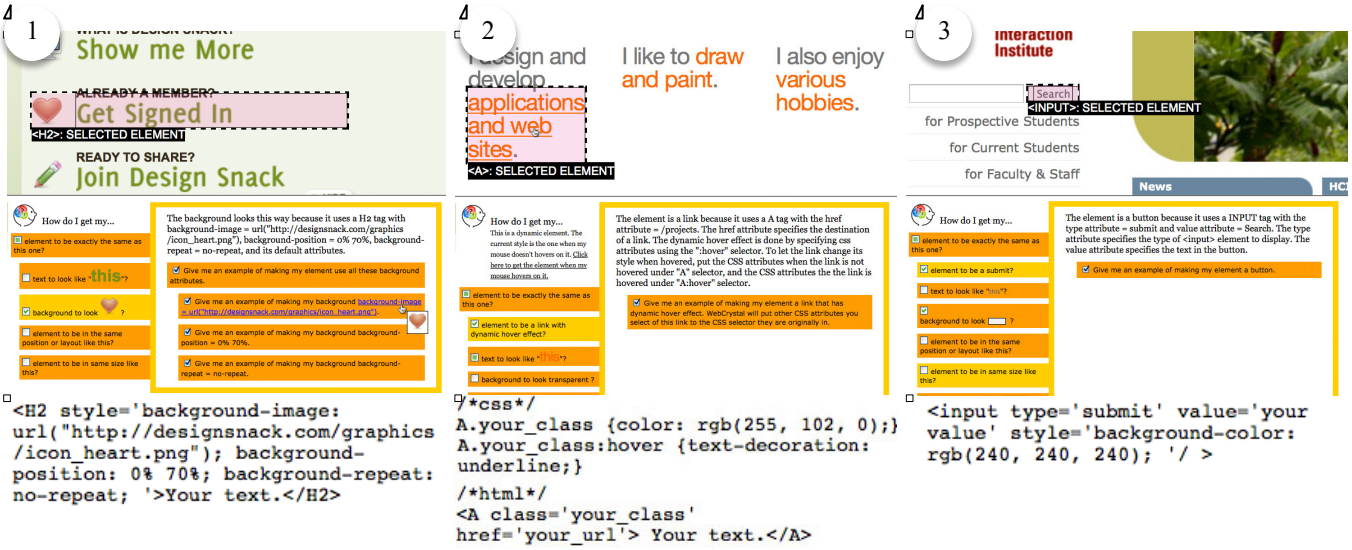

Figure 5: WebCrystal answers how to recreate (1) an image background, (2) an interactive link element, and (3) a search button, with the generated code snippets.

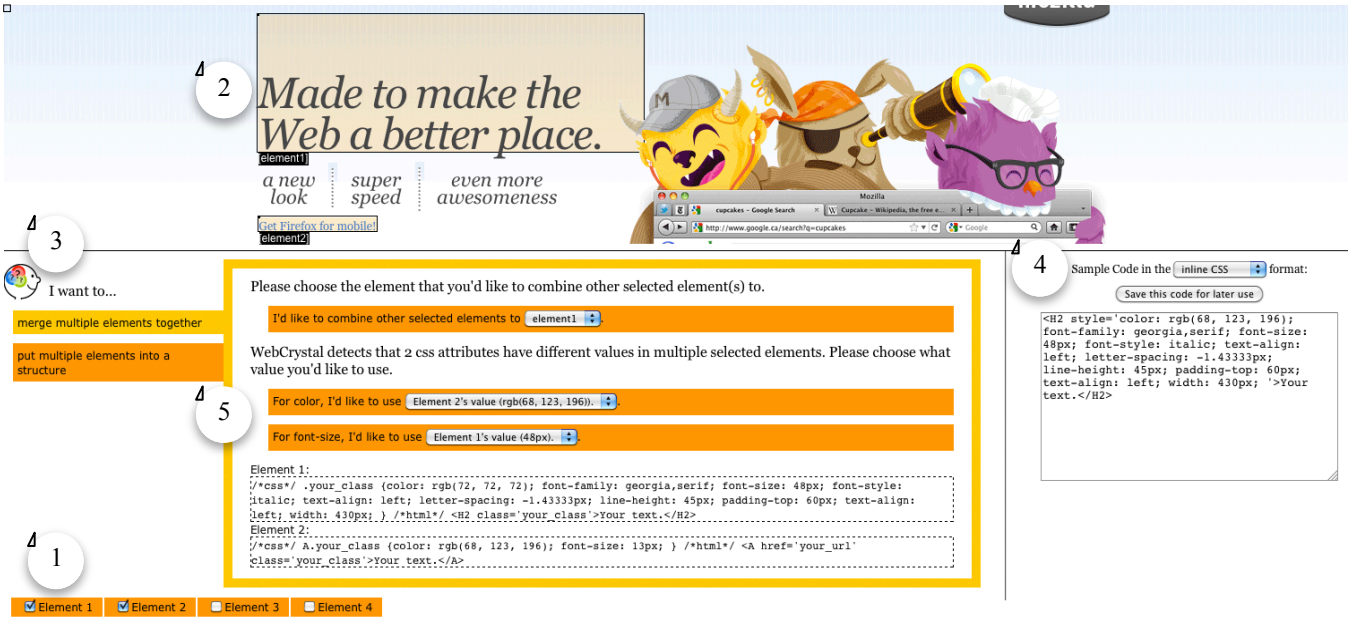

Figure 6: The interface for combining multiple elements. (1) the elements stored to be combined. (2) elements selected in (1) being highlighted in the web page in brown boxes. (3) the two commands of how to combine the elements. (4) the generated code for all selected elements merged together. (5) when an attribute is in multiple elements with conflicting values, the user can select which one to use.

even to select which specific aspects are desired from each element.

Another reason to investigate multiple web elements is to combine separate examples together. Inspired by Bricolage's approach of generating a new design by combining two web pages [13], WebCrystal supports creating a "style mashup" for users. For example, if the user likes the text style of one element, the background of another element and the size of a third element, WebCrystal can generate a code snippet for the user that creates a single element containing all of these desired design aspects.

The user interface for this feature is shown in Figures 2 and 6. First, the user selects the desired aspects of the first element in the usual way, which will cause the appropriate code for the first aspect to be displayed in the code pane. Then, the user selects the "save this code for later use" button (see Figure 2 at 5). The user then selects and shows the code for the other elements, and saves the code for each. The storing idea is inspired by the fact that designers in the real world often store the examples they like and retrieve them later in their design process [10].

Each saved code snippet is represented by a button with a system or user-defined name at the bottom of the WebCrystal window (see Figure 6 at 1). The user can select multiple snippets using the check boxes. The original element(s) from which the selected snippet(s) are copied from are highlighted on the web page in an orange semi-transparent box (Figure 6 at 2). Selecting multiple snippets takes users out from the question-asking interface to the "combining" interface, in which users give "I want to…" commands to the system to say how they want to combine the selected code (Figure 6 at 3). Clicking on "I want to merge multiple elements together" executes the "style mashup" feature, and WebCrystal generates code that has a single element with all the attributes in the selected code (Figure 6 at 4). If the saved snippets are elements of different types (e.g., one is a and another is a <li>), then a menu is generated to allow the user to select which type is desired in the code. If the same attribute has different values in different selected snippets (for example, if one snippet used the color gray and another used blue), WebCrystal will generate a menu for users to select which value they want (Figure 6 at 5).

The other top-level command in Figure 6 at 3 is "I want to put multiple elements into a structure." This is used when multiple items at different levels are desired to be combined into a multi-level structure in the result. For example, if one snippet is a styled list <ul> and another snippet is a styled list element <li>, and the desired result is a <ul> with the <li> inside of it. WebCrystal uses the hierarchical relation of the elements, and generates new code that has the appropriate code in the same hierarchy. Currently, WebCrystal knows how to combine elements that are siblings or that are parent and children into the same structure. In the future, we will investigate support for creating sensible structures from elements with no well-defined relations to each other, such as a <dt> with a <ul> or a with a <h1>.

USER STUDY

We evaluated the usefulness of WebCrystal in a small lab study. We polled a few web designers and web developers, and they reported that the most common ways that are used today to investigate code are the "View Source" menu item, along with tools such as Firebug [5] and the Chrome Developer Tool [4], which allow users to browse all of the HTML and CSS source code. Since WebCrystal is implemented as a Firefox plugin, we decided to compare it to Firebug in our user study.

Study Design

The study used a between-subject design. Participants were randomly assigned into two groups, the experimental group was given WebCrystal, and the control group was given Firebug [5]. In addition, both groups could use any of the standard FireFox features, such as View Source. Participants were also allowed to use any desired online resources to help with their tasks. In both conditions, the participants used the same special-purpose testing environment we created. This environment contains a text pane for entering the code, a "your output" pane that shows a preview of what a rendering of the code in the text pane will produce, and a "desired output" pane showing a preview of the correct answer (Figure 7). The participants' goal was to make the "your output" pane look and behave the same way as the "desired output" pane. Inside the text pane for each task, there was a small piece of code automatically inserted by the testing environment, for the participants to start from. Participants were told that they had to use this code as part of their answers. We did this to shrink the solution space and focus the task on just the reuse of the example code. We measured both the success rate and completion time on each task.

Participants

Both groups had 6 participants, all graduate students in our university. All participants had previous experience in writing HTML and CSS code. Participants rated their proficiency with both HTML and CSS language on a 4-point scale from "novice" to "superior". The average rating for HTML proficiency of all participants was 2.3 and for CSS was 2 out of 4. We also asked the participants to rate their level of experience with using Firebug or other web inspection tools on a 4-point scale of "none" to "expert". The average rating was 2.3. There were no significant differences in these measures between groups.

Tasks

All participants received the same 10 tasks in the same order. We designed the tasks to have 3 different levels of difficulty. The first 5 questions were the easiest, and were about reproducing the style of a single element. For example, one was to recreate a vertical separation line pointed to by the arrow in Figure 8 at 1. Next were 2 medium-difficulty questions, which were about reproducing the interactive behavior of an element. For example, one was to recreate a button that will change its style from green to red when hovered over by the mouse cursor like the one in Figure 8 at 2. The last 3 questions were the hardest, and were about reproducing the style of multiple elements. For example, recreating a same 3-grid structure like the one in Figure 8 at 3, but changing the color of the text to the blue color pointed by the arrow. The time limit for answering easy and medium tasks was 6 minutes and for hard tasks was 8 minutes. If users had not finished the task by the end of the time limit, they were marked as "uncompleted."

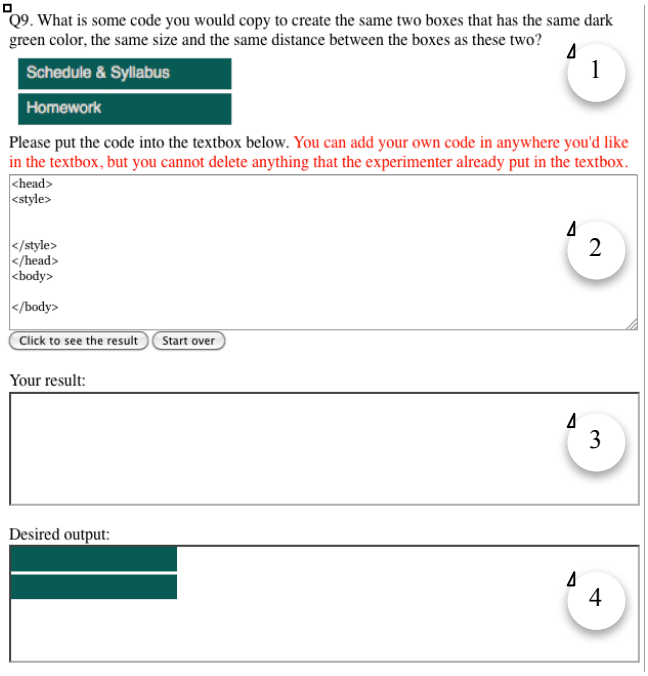

Figure 7: The testing environment contains (1) a question description, (2) a text pane for entering code. When clicking on the "Click to see the result" button, (3) will show a preview of (2). The correct output is shown at (4).

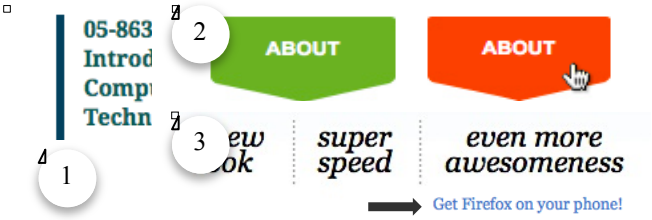

Figure 8: Tasks of 3 levels of difficulty.

Procedure

After giving consent to the study, the participants in both groups received a 15-minute tutorial on their tool (Firebug or WebCrystal) followed by a 5-minutes tutorial on the testing environment (Figure 7). The WebCrystal group was shown all the features described in the previous sections, and the Firebug group was shown all of its features that are relevant to HTML and CSS investigations, including view source, enabling and disabling CSS properties, and inspecting the CSS layout. We did not train participants in using

other Firebug features , such as network monitoring or JavaScript debugging, since they were irrelevant to the tasks.

After the tutorials, participants started to do the first task. For each task, the experimenter would read the task description and then after reading the whole description would start timing. Participants were told to inform the experimenter when they thought they had the correct answer. The experimenter then stopped the clock, checked their work, and told them if they gave a correct answer. If the answer was wrong, participants could choose to work on it some more or give up. However, in our study, none of the participants chose to give up on any of the tasks. All participants worked until the time ran out or they succeeded. After attempting all the tasks, the participants answered a short interview to provide feedback. Participants were paid $15 after the study.

Results

We analyzed the data using a random-effect logit model to predict task completion rate, and a random-effect linear model to predict task completion time, with all the observation of one person as a group. The results are presented in Figures 9 and 10. Participants in the WebCrystal group completed an average of 9.67 (97%) of the tasks, whereas participants in the Firebug group completed an average of 7.83 (78%). The difference in task completion rate between two groups is significant (coef. = 2.49, p=.013).

We also analyzed the average time per task for all those participants who completed successfully. Participants using WebCrystal spent an average of 94.27 seconds (SD = 75.29) on tasks, whereas participants using Firebug spent 115.09 seconds (SD = 98.26). The difference is significant (coef. = -30.47, p = .036). As shown in Figure 10, the difference increases a bit as the tasks get more difficult. Taking the ”hard” tasks alone, successful participants were 43% faster with WebCrystal (147.91 seconds vs 211.91 seconds).

The user study confirmed that our difficulty ratings were valid since across both groups, the main effect of task difficulty was significant on both task completion rate (coef. = -1.42, p < .001) and task completion time (coef. = 53.58, p < .001).

In the interviews, participants in the WebCrystal group expressed a great interest in using WebCrystal in real life:

“This is cool… are you gonna release it?”
“Can I have it?”
“Do you have this in Chrome?”
“This is definitely useful… I’ll pay a little money for it.”

Among the 6 WebCrystal users, 5 of them asked about the tool’s availability, and 3 of them volunteered to help on testing and reporting bugs after its release. This is particularly encouraging since most of our participants have previous experience in using other web development tools.

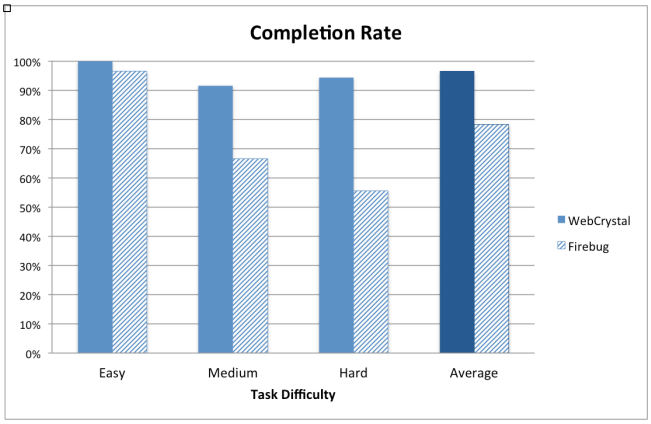

Figure 9: The completion rate of two groups in three levels of difficulty tasks and the overall average. Taller bars are better.

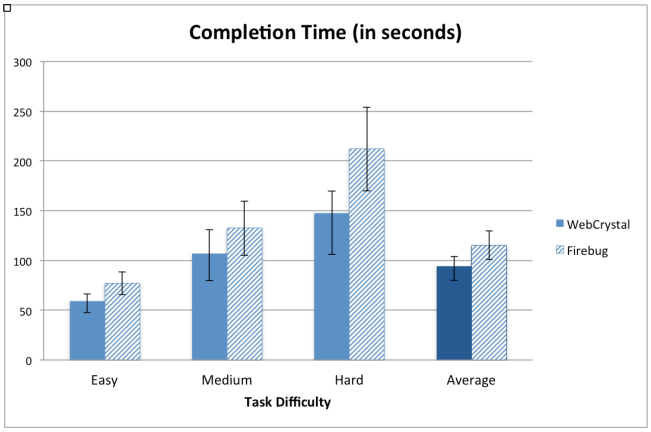

Figure 10: For the participants who completed the tasks, the average time they took in 3 levels of difficulty, and the overall average with standard errors. Shorter bars are better.

DISCUSSION

The key to successfully finishing the tasks in our user study is to quickly identify all of the required attributes and combine them to form usable code, while avoiding including inappropriate attributes. This is especially critical with the difficult tasks because they involve more attributes. We feel this study has external validity since reproducing a design aspect in real life almost always requires copying multiple related attributes. For example, to recreate text that has the same style as in Figure 2 at 1 requires users to specify 6 different text attributes. WebCrystal users took advantage of the way that attributes are already classified into different categories. They accessed the appropriate attributes by selecting different questions to ask the system to show them what they wanted. In contrast, the Firebug users were presented with all of the attributes by the system and needed to determine on their own which ones were related to the task.

When identifying an individual attribute, WebCrystal users viewed the rendered style of the attribute in each checkbox description to understand the effect of that attribute. Participants reported that this “graphical indicator” of an attribute

and its effect were very helpful, because they actively provided the relevant information. This led the users to ask the right questions. On the other hand, Firebug users used the live editing feature in Firebug to turn the attributes on and off to view the changes in the web page. Although this feature also successfully explained the effect of an attribute to the user, it required the users to be the initiator of the inquiry process, and since it modifies the example page, it can interfere with appropriate rendering. In other words, with Firefox, all the attributes were passively waiting to be explored, and users had to first guess or know which attribute could be relevant and then decide if they wanted to use the live editing feature to check if it was truly the one they needed. We observed much trial-and-error clicking since users' first guesses were often not correct. We also found that for some of the tasks which Firebug participants failed to complete, there were attributes which participants never explored because they incorrectly assumed these attributes were irrelevant.

We observed that all WebCrystal users benefitted from the customized, ready-to-use code snippet generated by the system when doing both easy and hard tasks. Participants in the WebCrystal group liked using the checkboxes to select attributes to include in the code snippet, and thought it was very easy and efficient to use. Copying and pasting the generated code snippet prevented users from having any potential syntax errors and typos. In the Firebug group, some users chose to retype the attributes they thought were relevant rather than copying and pasting. This is because attributes are separated into different lines in Firebug, which would therefore require users to perform multiple selecting, copying and pasting operations. The result of this retyping was that more typos occurred, which users often did not notice at first. The typos often caused incorrect output, which made the users think that they had identified the wrong attribute. Then, instead of checking their code, users went searching in the example code again and got more and more confused, until they finally discovered that it was the typo that caused the wrong output. Conventional web editors such as Dreamweaver have the ability to check syntax errors for users, but still may not identify if there is a typo in an attribute name or value. These observations suggest that having a customized, ready-to-use code snippet extracted from the example file not only saves the users' time but also might result in higher-quality code.

When performing the tasks, we often observed very exploratory usage of sample code by the WebCrystal users. In WebCrystal, because selecting all attributes under a category was so easy (simply by checking one checkbox), when participants where not sure about the effect of individual attributes, some would just check everything that related to the task description, and see how the resulting code snippet worked using the preview pane in the answering environment. We observed fewer attempts like this in the Firebug group, because both identifying the relevant attributes and copying them were time-consuming.

Both WebCrystal and Firebug users considered the tool they used to be very helpful for their tasks. Some participants with advanced knowledge of HTML and CSS expressed that the textual explanations WebCrystal provided were less useful for them because they already knew most of the attribute names and effects. The main performance difference in completion time between expert users in the two groups seemed to be caused by the fast copy-and-paste ability in WebCrystal, which saved them from typing (and typos) and kept track of the attributes one by one. For novice and intermediate HTML and CSS users, WebCrystal's textual descriptions seemed to be more useful. Both expert and novice participants reported that they liked the question-asking style of the interface. They thought it was very intuitive to use and easy to learn. While novice and intermediate participants in the Firebug group struggled to identify the right attributes to use and even to form syntactically correct code, novice and intermediate participants in the WebCrystal group found the correct attributes by asking a higher-level question and directly copying the ready-to-use code as their answer.

LIMITATIONS

WebCrystal focuses on explaining HTML and CSS, and does not handle JavaScript, Flash, or other scripting languages for interactive behaviors. WebCrystal currently flattens any CSS inheritance in the example, and generates code that contains all the required attribute values together. Our motivation for this was to help users quickly reproduce a desired element by directly copying the example code into their own web page. The downside of this piece-by-piece styling approach is that it does not take advantage of the cascading nature of CSS, and might result in a difficult-to-maintain file as the number of copied elements become larger. Also, the style of an element is affected by its parents because some of the attributes can be inherited. Therefore, we cannot be sure whether the example code pasted from WebCrystal will work exactly the same in the target web page as in the example without knowing the hierarchical structure of where it is pasted. As a simple example, WebCrystal leaves out attributes which have their default value in the source file, and these attributes may have different default values in the target file. This could be addressed by integrating WebCrystal with a web editor to support the user's choice of whether to override or inherit any attributes that differ in the example and target files.

CONCLUSIONS AND FUTURE WORK

WebCrystal is a tool that helps users to understand how an example is constructed and to reproduce the example in their own web page. Its interaction techniques based on asking and answering questions proved effective, easy-to-learn and well-liked by both novice and experienced web developers. WebCrystal successfully allowed fast copying-and-pasting of desired attributes, and the storing and combining of attributes from multiple examples. The ability to specify the desired attributes and have the tool generate appropriate combined code for them proved to be an im-

portant advantage, compared to requiring users to combine the code by hand. WebCrystal is now available at http://www.cs.cmu.edu/~webcrystal/.

Future work could be in many directions. One is to integrate WebCrystal with web editing tools such as Dreamweaver or Eclipse to facilitate intelligent pasting of the example code in a way that would consider the hierarchy relations among elements, as described in the previous sections.

Another direction is to extend the system's ability to be able to explain the construction of interactive behaviors of a webpage. A previous system, FireCrystal [19], lets its users playback the interactions with web pages using a timeline and displays relevant code. From our user study, we observed that participants benefitted from selecting and augmenting desired code snippets through a hierarchical question-asking interface. Combining the interaction techniques in WebCrystal and FireCrystal, one could imagine a system that records interactions in a web page, divides complex interactions into smaller and easily understandable parts, and lets users access relevant code of each part by asking hierarchical questions.

Finally, we are also interested in observing web developers using WebCrystal for real-world tasks. What elements are most web developers interested in? What are the most common questions that web developers ask when recreating an element? How does WebCrystal affect on the design process? Understanding these questions would give us more insights on designing future tools that support reusing examples in web authoring.

ACKNOWLEDGMENTS

We would like to thank Ruogu Kang, Yanjin Long, Haiyi Zhu and Colleen Stuart for their help with the statistics for the paper, and Andrew Faulring for his help with the user study. This research was funded in part by the NSF under grant IIS-1116724. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the NSF.

REFERENCES

1. Brandt, J., Guo, P. J., Lewenstein, J., Dontcheva, M., Klemmer, S. R. Two studies of opportunistic programming: interleaving web foraging, learning, and writing code. *Proc. CHI* (2009), ACM, pp. 1589-1598.
2. Brandt, J., Pattamatta, V., Choi, W., Hsieh, B., Klemmer, S. R. *Rehearse: Helping Programmers Adapt Examples by Visualizing Execution and Highlighting Related*. Tech. rep. CSTR 2010-05, Stanford, Oct. 2010.
3. Buxton, B. *Sketching User Experiences*. Morgan Kaufmann, 2007.
4. *Chrome Developer Tools*. http://code.google.com/chrome/devtools/
5. *Firebug.* http://getfirebug.com/
6. Fitzgerald, M. CopyStyler: Web design by example. Master Thesis, Dept. of EECS, MIT, May 2008.
7. Gibson, D., Punera, K., Tomkins, A. The volume and evolution of Web page templates. *Proc. WWW* (2005), ACM, pp. 830-839.
8. *Graphic Mania.* http://www.graphicmania.net/are-web-designers-required-to-know-how-to-code/
9. Hashimoto, Y., Igarashi, T. Retrieving web page layouts usingsketches to support example-based web design. 2nd Eurographics Workshop on Sketch-Based Interfaces and Modeling (2005).
10. Herring, S. R., Chang, C.-C., Krantzler, J., Bailey, B. P. Getting inspired!: understanding how and why examples are used in creative design practice. *Proc. CHI* (2009), ACM, pp. 87-96.
11. *jQuery.* http://jquery.com/
12. Ko, A. J., Myers, B. A. Finding Causes of Program Output with the Java Whyline. *Proc. CHI* (2009), ACM, pp. 1569-1578.
13. Kumar, R., Talton, J. O., Ahmad, A., Klemmer, S. R. Bricolage: Example-Based Retargeting for Web Design. *Proc. CHI (2011)*, ACM, pp. 2197-2206.
14. Lee, B., Srivastava, S., Kumar, R., Brafman, R., Klemmer, S. R. Designing with interactive example galleries. *Proc. CHI* (2010), ACM, pp. 2257-2266.
15. Marks, J. et al. Design galleries: a general approach to setting parameters for computer graphics and animation. *Proc. SIGGRAPH* (1997), ACM, pp. 389-400.
16. *MDC Doc Center.* https://developer.mozilla.org/en/xul
17. Myers, B. A., Weitzman, D. A., Ko, A. J., Chau, D. H. Answering why and why not questions in user interfaces. *Proc. CHI* (2006), ACM, pp. 397-406.
18. Nardi, B. *A small matter of programming*. MIT Press. 1993.
19. Oney, S., Myers, B. A. FireCrystal: Understanding Interactive Behaviors in Dynamic Web Pages. *IEEE Symp. On VL/HCC* (2009), pp. 105-108.
20. Ritchie, D., Kejriwal, A. A., Klemmer, S. R. d.tour: style-based exploration of design example galleries. *Proc. UIST* (2011), ACM, pp. 165-174.