Towards a Taxonomy of Errors in HTML and CSS

Thomas H. Park, Ankur Saxena, Swathi Jagannath, Susan Wiedenbeck, Andrea Forte
College of Information Science and Technology
Drexel University
3141 Chestnut Street
Philadelphia, PA 19104, USA
{thomas.park, ankur.saxena, swathi.jagannath, susan.wiedenbeck, aforte}@drexel.edu

ABSTRACT
As part of a larger research agenda to explore web development as a context for learning computational literacy skills, we investigate errors people make while writing code in HTML and CSS. We report on a lab-based study in which 20 participants were video recorded as they completed coding tasks. We have applied the skills-rules-knowledge framework to segment this data by the cognitive causes of errors they made, and present a taxonomy of these errors. Our findings demonstrate how the skills-rules-framework can be used to analyze coding errors, provide insight about the origins of these errors, and suggest ways that the design of web development tools can be improved to support learning and practice with HTML and CSS.

Categories and Subject Descriptors
K.3.2. [**Computers and Education**]: Computers and Information Science Education—*computer science education*

General Terms
Design, Human Factors

Keywords
Computing education, errors, web development

1. INTRODUCTION
Building web pages is not easy. Although markup languages may not be considered as computationally expressive as many other languages, Blackwell notes that they possess many of the pitfalls of programming: "even the abstractions of HTML provide the opportunity for syntax errors, runtime errors, or bugs in the form of unintended or exceptional behaviors" [2]. Beginners who learn HTML and CSS encounter many opportunities to learn from the process of authoring code for a computer to interpret, making mistakes along the way, and recovering from those mistakes.

Much research examines the difficulties novices have learning to program, with the goal of better supporting learners; however, the errors people make while writing HTML and CSS are largely unexamined. We view this as an important gap in the literature. For many people, HTML and CSS provide a first exposure to creative computation. Insurmountable difficulties can discourage beginners, while bad habits and misconceptions here may limit what can be learned from such an experience. On the other hand, positive, productive experiences can serve as a stepping-stone to sustained and deepening engagement with computing [15].

In this study, we seek to identify the errors people make while writing code in HTML and CSS, and examine the cognitive origins of these errors. These findings are instrumental in our continuing efforts to design a web editor for beginners that is a pedagogically superior alternative to existing tools.

The balance of our paper is organized as follows. Section 2 reviews prior research on programming errors and web development. Section 3 describes our task study and our use of the skills-rules-knowledge framework to analyze our data. Section 4 details three examples of the coding errors we observed, and presents a taxonomy of HTML and CSS errors. Finally, section 5 discusses our findings in relation to education and designing systems to support beginners.

2. RELATED WORK
There is a large body of literature that examines and describes the kinds of coding errors people make, their strategies for recovering from such errors, and the role that errors play in learning to code. As Spohrer and Soloway [16] observed, "All bugs are not created equal. Some bugs occur over and over again in novices' programs, while others occur rarely."

Spoher and Soloway's remark was based on a study of syntactically correct programs written by students in Pascal, where the researchers cataloged 101 different bug types and found that 10 percent of bug types accounted for between 32 and 46 percent of observed bug instances. They found that despite the conventional wisdom that most bugs are due to misconceptions about the semantics of language constructs, the majority arose when the students encountered boundary conditions or interactions between different pieces of code. Anderson and Jeffries [1] confirmed that even increasing the complexity of *irrelevant* aspects of a programming problem leads to more errors. In their study of students programming in LISP, they found that these errors stem mostly from slips such as forgetting parentheses, rather than enduring misconceptions.

Turning their attention to errors children make using a natural-language-style programming language, Bruckman and Edwards [4] offered their own classification scheme organized into seven categories. According to this scheme, they found that most errors involve object manipulation, command-line syntax, and typos. Youngs provided a broader investigation into programming errors by classifying errors made in a variety of languages by 42 programmers, both novice and expert, in terms of statement type (e.g., assignment, iteration), depth of understanding needed to correct the error (e.g., syntax, semantic, logic), the manifestation of the error (e.g., formatting, omission,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ICER'13, August 12–14, 2013, San Diego, California, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2243-0 /13/08…$15.00.
http://dx.doi.org/10.1145/2493394.2493405

illegal operation), and the system response [18]. He reported that beginners struggle most with semantic errors, while syntax, semantic, and logical errors occur in roughly equal proportion for experts. Finally, Ko and Myers have conducted an exhaustive review of studies of programming errors in [7].

Errors have also been studied in programming more broadly defined. For instance, Panko [10] reviewed 13 studies of errors people make while developing spreadsheets, and proposed a taxonomy of quantitative errors, which give an incorrect result, and qualitative errors, which are broken down further into mechanical errors such as mistyping a number, logic errors that result from a mistake in reasoning, and omission errors where something is left out. In the reviewed studies, all error types were made regularly, although logic and omission errors occurred more frequently and were more difficult to detect by spreadsheet developers than mechanical errors.

The domain of web development has received less attention from researchers. Désilets et al. [5] observed wiki usage in grade 4 classes and described children's frequent errors with the syntax and semantics of hyperlinks. Miller et al. [9] presented a classification of the errors college students made in referencing resource files such as a link or image when coding webpages. Rather than focusing on errors, Dorn and Guzdial [6] employed card-sorting tasks to assess the programming knowledge of professional web developers.

Park and Wiedenbeck used content analysis to examine online forums for an introductory web development course, in order to understand what concepts and tasks proved difficult for beginner web developers [12]. This work revealed that early barriers to successful web coding experiences are diverse: problems arising from writing and testing code were most common (34 percent); however difficulties related to the instructional and technological environment were not far behind (30 and 25 percent respectively).

The findings from [12] inspired us to begin work on a web editor specially designed for learners. However, despite the wealth of literature on programming errors, we lacked a detailed understanding of the errors people are likely to make when attempting to write HTML and CSS. Moreover, we had ideas about scaffolding that might support learners, but lacked empirically grounded accounts of what causes people to make errors and how such errors could be productively overcome. This led us to ask:

- What errors do people make when constructing web pages in HTML and CSS?
- What are the sources of these errors?
- Once made, how well do people recover from such errors?

3. METHODS

In order to collect the detailed observations necessary to understand the errors people make when constructing HTML and CSS and why, we conducted a lab-based study. We observed and recorded 20 participants as they completed a set of HTML and CSS coding tasks using a think-aloud protocol. We then used open and axial coding processes to analyze video and screen capture data to construct a taxonomy of errors.

3.1 Participants

To capture as broad a sample of errors as possible, we sought participants with a wide range of experience in HTML and CSS and did not exclude any background or profession. We used a variety of recruitment tactics from announcements in beginner web development classes, to flyers posted on university campuses, to a classified ad in the web design section of Craigslist. Participants were offered $20 for their time.

A total of 20 people, 7 female and 13 male, took part in the study. Their ages ranged from 18 to 47 (M=24.4) and their backgrounds included digital media, environmental science, business, and art. The two participants who indicated web design as their profession stated that they relied on content management systems like Wordpress to do their work, and did not practice a great deal of coding. In addition to HTML and CSS, 17 of the 20 participants reported some programming experience. The participants are described more fully in Table 5.

3.2 Protocol

In order to provide a consistent experience for all participants and to record the sessions, participants were invited to our usability lab and asked to complete a set of five coding tasks involving HTML and CSS. The tasks were preceded with a questionnaire and brief interview that collected information on demographics and prior experience. For example, participants were asked to rate their own expertise with HTML, CSS, and any programming languages as no experience (0), beginner (1), intermediate (2), or advanced (3). As in the earlier studies of programming errors, we expected expertise to be a significant factor in their outcomes.

Participants used the first iteration of our web editor, openHTML, to complete the tasks (see Figure 1). Our design strategy is to begin with the simplest possible environment and use an iterative approach to extend it with added functionality as we learn about what learners need and want [11]. This approach made the first version of openHTML an ideal environment for the study since the editor is as simple as we could make it, lacking the bells and whistles of more complex editors. Moreover, all participants were equally unfamiliar with the tool. Participants were given an orientation to openHTML before the study began.

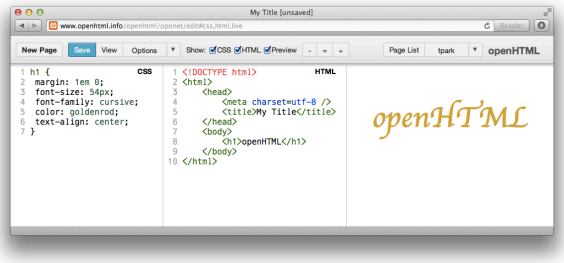

Figure 1. openHTML, the web editor used in the study. The interface displays panes CSS, HTML, and live preview.

For each coding task, we gave participants printed instructions containing multiple sub-goals as well as an image depicting the expected output of the rendered web page. We asked them to complete tasks to the best of their ability using whatever resources they would normally use, including web searches. We explained the think aloud protocol and encouraged participants to vocalize their thought processes as they completed the tasks. A maximum of 30 minutes was provided for each task, and participants were allowed to end a task at any time. After each task, we asked follow-up questions to clarify their understanding and intent. At the end of each session, we asked a series of follow-up questions designed to probe understanding of basic computational concepts. Sessions were video recorded; participants averaged

approximately 38 minutes of coding activity, totaling over 12 hours of video data overall.

3.3 Participant Tasks

Participants completed 5 tasks that involved writing or modifying HTML and CSS. We piloted the tasks to ensure that participants could complete them in 10 to 15 minutes. The tasks were also designed to provide broad coverage of HTML and CSS constructs, setting a low floor and steadily increasing in sophistication. For all of the tasks, the HTML pane was seeded with boilerplate code for the HTML5 document type declaration and `html`, `head`, `title`, `meta charset`, and `body` tags; additional code was seeded for Task 3 requiring the code to be extended, and Task 4 requiring three bugs to be fixed. The tasks are summarized in Table 1.

Table 1. The coding tasks.

Task	Requirements
1	a) Create a heading b) Create a paragraph c) Create an ordered list d) Create an ordered sub-list
2	a) Embed a hyperlink b) Embed an image c) Make the image into a hyperlink
3	a) Center the text alignment in the provided table b) Set the background color of the rows with "pro" text to green and the "con" text to red c) Color the provided hyperlink green d) Color the provided hyperlink red on hover event
4	a) Find and fix bug 1: broken image b) Find and fix bug 2: unclosed tag c) Find and fix bug 3: unmatched CSS selector
5	a) Create a container div b) Position the container div in the center horizontally c) Create a sidebar div d) Position the sidebar div inside the container div on the right

3.4 Data Analysis

Two researchers coded video data in three iterative rounds using the software Morae. The researchers did not apply a pre-determined codebook; rather, the goal was to use the coding exercise as a way of developing an inventory of errors. As the two coders worked together, they reconciled disagreement through further discussion and, in this way, converged on a shared conceptual vocabulary.

In the first round of coding, every occurrence of an error was marked. In alignment with Youngs' definition of programming errors [18], we defined errors as code written by the participant with invalid syntax, or that resulted in actual or potential output (webpage rendering) that was not desirable according to the task or the participant's interpretation of that task. A total of 791 errors were identified. In this initial round, we immediately classified 463 errors as being typographical in nature and immediately resolved (e.g., misspelling a word and then correcting it without a substantial time delay or shift in attention), since they were deemed trivial and constituted a majority of instances.

Table 2. The coding scheme for errors.

Code	Values
Level	skill, rule, knowledge
Type	typo, obsolete construct, css selectors, etc.
Resolution	resolved, unresolved, bypassed

In the next round of coding, we classified the remaining 328 errors based on our emergent coding scheme, which was informed by the skills-rules-knowledge framework, a hierarchical model of human behavior organized in terms of cognitive effort [13]. Reason offers a thorough treatment of the skills-rules-knowledge framework in [14], which helped us consider the type of cognitive breakdowns at the root of each error:

- Skill-based behaviors, such as typing, are "sensory-motor performance[s] tak[ing] place without conscious control as smooth, automated, and highly integrated patterns of behavior." Errors at this level are the result of unintended actions from physical slips, inattention, or mode confusion.
- Rule-based behaviors are comprised of "a sequence of subroutines in a familiar work situation… typically controlled by a stored rule or procedure." Rule-based behavior is guided by conscious and goal-oriented planning. Errors here result from intentional actions driven by the application of bad rules or the misapplication of previously good rules to exceptional circumstances.
- Knowledge-based behaviors occur at a higher conceptual level when a person faces an unfamiliar situation that necessitates ad-hoc experimentation and problem solving. Errors at this level, or more aptly "breakdowns," result from an incomplete or inaccurate understanding of the situation. Typically, multiple errors are made in succession, entwined with experimentation and information searches.

We assigned each of the errors to one of these three levels. In order to make this assignment, we relied not only on observed coding behavior but other cues, including the participants' verbalizations while coding, their reactions when errors were detected and resolved, and, importantly, their strategies for resolving them. For instance, a web search could be used to remember complicated syntax, suggesting rule-based behavior, or for just-in-time learning of a broader topic [3], typical for trying to address a knowledge-based breakdown. Table 3 outlines the heuristics that we applied during this part of coding.

Table 3. Heuristics based on [14] used to classify errors as occurring at the skill, rule, or knowledge-based levels of performance.

	Skill	Rule	Knowledge
Types of Activity	Quick, routine actions	Simple if-then rules	Slow, information seeking
Control Mode	Mainly by automatic processes	Mainly by automatic processes	Limited, conscious processes
Perception	Feedforward	Feedforward	Feedback
Intention	Unintended actions	Intended actions	Intended actions
Solution	Indicator of existence	Brief explanation	Extensive learning

We developed a detailed taxonomy of error types at each of the three levels through an inductive, data-driven process. At the skill-based level, errors tended to be simple, such as forgetting to type a semicolon. At the rule-based level, errors became more complex, for example using an attribute that has been deprecated. Knowledge-based level errors proved to be the most complex, such as a complete lack of understanding of the positioning model, which determines how elements are laid out in relation to each other on the web page. We also coded whether errors were ultimately resolved, unresolved, or bypassed in favor of a different approach. In the second and third rounds of analysis, we reviewed the codes and made refinements where needed.

4. FINDINGS

Participants averaged 39.6 errors per session (including all tasks) (SD=15.0), ranging from 15 to 63. Breaking down completion time and error count by task (Table 4) reveals a rough trend of increasing time and errors, although as we will see in later sections, not all errors are created equal. Task 4 required fixing existing code rather than writing new code, which may partially explain the low completion times and error counts.

Table 4. Mean task completion time in minutes and error count for each task.

Task	1	2	3	4	5
Time (SD)	5.42 (4.61)	5.94 (3.96)	9.40 (5.56)	6.51 (4.62)	10.95 (5.69)
Errors (SD)	7.55 (4.75)	6.70 (4.26)	7.85 (5.44)	4.20 (4.12)	13.25 (8.16)

In this section, we describe three cases in detail, representing errors at the skill-based, rule-based, and knowledge-based level. We then present a taxonomy of HTML and CSS errors based on all of the errors we observed.

4.1 A Tale of Three Errors

4.1.1 Skill-Based Error

Participant 15, a 41-year-old web designer, is working on embedding an image in Task 2, which instructs that he include an `alt` attribute that specifies alternate text when the image cannot be found. The correct code should resemble the following:

```
<img src="http://constitutioncenter.org/
images/ui/logo-ncc.gif" alt="My Image" />
```

However, Participant 15 forgets the opening quote in the `alt` attribute's value.

```
<img src="http://constitutioncenter.org/
images/ui/logo-ncc.gif" alt=My Image" />
```

After examining the code for a minute, he finally spots the source of the error, exclaiming, "Oh! That's it," before fixing it. Despite successfully enclosing values with quotes numerous times before and after this instance, he makes a skill-based error here, whether due to cognitive overload, inattention, or a slip of the finger. In this case, merely signaling the presence of the missing quotation error would be sufficient information to fix it.

Table 5. Summary of participants, with expertise in HTML, CSS, and most familiar programming language on a scale of 0 to 3, and skill-based, rule-based, knowledge-based, and total error counts.

P	Gender	Age	Current Profession	HTML	CSS	Prog	S	R	K	Total	Unsolved
1	Female	19	Student (Digital Media)	••	••	•	38	3	0	**41**	**2.4%**
2	Female	20	Student (Digital Media)	••	••	••	21	6	2	**29**	**34.5%**
3	Male	20	Student (Computer Science)	••	••	•••	47	8	3	**58**	**1.7%**
4	Male	20	Student (Business)	•••	•••	•	39	23	0	**62**	**16.1%**
5	Male	19	Student (Information Systems)	••	•	•	21	6	7	**34**	**17.7%**
6	Male	25	Student (Information Science)	••	••	•	20	16	8	**44**	**38.6%**
7	Female	22	Student (Digital Media)	••	••	•	36	3	1	**40**	**2.5%**
8	Male	23	Visual Effects Art	•••	•••	••	21	0	0	**21**	**4.8%**
9	Male	23	Student (Digital Media)	•••	•••	•••	56	6	1	**63**	**6.4%**
10	Male	20	Student (CS)	•••	•••	•••	16	8	1	**25**	**4.0%**
11	Female	29	Student (Environmental Science)	•	•	•	39	6	12	**57**	**7.0%**
12	Male	20	Student (Information Systems)	•••	•••	•••	23	3	0	**26**	**3.9%**
13	Male	36	Law	•		•	18	3	11	**32**	**28.1%**
14	Male	22	Student (Information Technology)	•	•	•	9	2	4	**15**	**26.7%**
15	Male	41	Web Design	•	••	••	37	8	10	**55**	**21.8%**
16	Female	19	Student (Art)	•	•		26	13	5	**44**	**27.3%**
17	Female	47	Web Design	•	•		26	3	5	**34**	**17.7%**
18	Male	21	Student (Business)	•••	•••	••	22	2	1	**25**	**8.0%**
19	Female	24	Student (Education)	•	•		14	8	5	**27**	**33.3%**
20	Male	18	Student (Business)	•	•	•	32	7	20	**59**	**30.5%**

4.1.2 Rule-Based Error

Participant 5, a 19-year-old college student, is progressing with Task 5, which requires him to create multiple `div` elements in HTML and style them using CSS. To this end, he assigns the elements classes in HTML and selects those classes in CSS. These are skills that he successfully used earlier to complete Task 4.

He sets the class of one div to "2" and assigns the class a blue background color. To his surprise, the div does not change color. Though he does not realize it, the cause of this error is that class names cannot begin with a number.

This episode is illustrative of rule-based errors. Participant 5 is familiar with the general rule for how to set classes in HTML, and how to select them in CSS. But he comes up against an unfamiliar exception in how classes can be named. Although he is able to overcome this, he expends significant time and effort to do so, and in the end may still not fully comprehend the source of the error. In this case, the simple elaboration of a known rule is likely sufficient for resolving the error.

4.1.3 Knowledge-Based Error

In Task 3, Participant 20 is asked to style the text in each cell of the provided table by aligning it to the right. He begins by opening up a website he used in an earlier task to reference the syntax of common tags. On the website is a section called "Alignment tags," which includes the following deprecated code for aligning text to the right.

```
<P ALIGN=Right>your text
```

He copies the code, pastes it into his own, and modifies it to create the following:

```
<table><ALIGN=Right>
    <tr><td>Pro: Low Unemployment</td></tr>
```

Observing that this code doesn't have the desired effect, he tinkers with the placement of the align code, moving it inside the `td` element without any success. He moves it again, this time between `tr` and `td` tags. It still doesn't work.

He searches the web with a query for "align right table". The top result is a question and answer site, where he spots code using the align attribute:

```
<tr><td>..</td><td align='right'>10.00</td></tr>
```

He copies and pastes part of this HTML snippet into the CSS pane, resulting in the following code.

```
table {
    align='right'
}
```

The style is still not taking effect, so Participant 21 spends the next minute carefully inspecting his code. He adds dummy text between the `tr` and `td` tags, confirming that it has some effect on the live preview before quickly deleting it. Next, he conducts another query for "css align right table" and scans three different pages. He comments to the researcher, as he points to the code he had added to the CSS pane, "It said to put this in here. Almost exactly like that." He continues with several more web searches, using general queries like "using css" and "apply css attribute". After much tinkering with the code, Participant 21 gives up six minutes after he started moves on to the next part of the task.

Participant 21's struggles with Task 3 involved the fundamentals of HTML and CSS, and are representative of errors at the knowledge-based level. He has significant knowledge gaps in the structure of an HTML tag, demonstrates persistent confusion between HTML and CSS code, and engages in lengthy web searches. At this level, resolution requires substantial learning.

4.2 Classifying the Errors

To produce a robust classification of errors, we examined not only the errors themselves, but also the context and response to the errors in a process similar to axial coding from grounded theory [17] and informed by our understanding of errors as driven by skills, rules, or knowledge deficits. This yielded a unique set of codes at each level of the skills-rules-knowledge framework. Through our analysis, we found that 70.9 percent of errors occurred at the skill-based, 16.9 percent at the rule-based, and 12.1 percent at the knowledge-based levels. A scant 4.3 percent of skill-based errors were unresolved, while 39.6 percent of rule-based and 52.1 percent of knowledge-based remained so. Tables 7 to 9 provide a description and example for each type, and a count of total and unresolved occurrences.

At the skill-based level, errors are caused by unintentional actions, such as a mental or physical slip, during highly routine activities. We found six major error types here, including typographical errors, forgetting to close paired constructs, missing a delimiter, accidentally mixing HTML and CSS syntax due to mode switches, confusing semantically similar constructs such as titles and headers, and placing code in a location other than intended. These errors do not always occur at the skill-based level, and so we had to rely on additional cues, such as their reaction to spotting an error and their attempts at fixing it, to classify them. Far and away, typographical errors were the most common, although nearly all of them were resolved. This was generally true for all skill-based errors.

At the rule-based level, errors also occur during relatively routine activities, but are caused by the intentional and consistent, but faulty, application of familiar rules. We found rule-based errors to be most diverse in their types. This makes sense given that they occur when encountering edge cases, where more general and previously reliable rules start to break down. The most common rule-based errors involved using the wrong name for a property or attribute, using an obsolete construct, and dealing with lists. Especially at this level, the error types are not meant to be comprehensive, but simply representative of the errors we observed in our study. We expect that countless others can be added to this list, and that this list is likely to change as standards evolve.

At the knowledge-based level, breakdowns are caused by a severe lack or misapprehension of relevant knowledge while facing an unfamiliar problem. Knowledge-based errors make up only a few types (Table 9), but they are central models governing HTML and CSS, broadly integrating many topics. HTML fundamentals and CSS fundamentals were most common, perhaps reflecting the expertise of participants and the nature of the tasks.

Table 6. Skill-Based Error Types.

Error Type	Description	Examples	Total	Unresolved

Typographical Errors	Physical slips in the typing process, as with tags, properties, and values	`</blcokquote>`
		`bacground-color`
		`width: 100ps;`
			495	**7**

Unclosed Pairs	Forgetting to close paired constructs or characters, such as tags, quotes,or braces	`<h1>Note`
		`<img src="foo.png>`
		`a { color: red;`
			27	**15**

Missing Delimiter	Forgetting other symbols that delimit data, such as semicolons in CSS rules and the hash symbol in hex values	`h1 {`
		`font-size: 20px`
		`color: 0000FF;`
		`}`
			6	**1**

Mixed Mode	Accidentally applying HTML syntax to CSS, or vice versa	`div { color=blue; }` `<div color=red;>`	**12**	**1**

Confused Similar Constructs	Mixing up semantically similar constructs	`title & h1`
		`color & background-color`
		`class & ID.`
			17	**0**

Misplaced Code	Accidentally pasting code or typing in the wrong location	`<a href=""http://foo.com></a>`	**4**	**0**
			561	**24**

Table 7. Rule-Based Error Types.

Error Type	Description	Examples	Total	Unresolved
Obsolete Construct	Using elements, attributes, and properties that once were valid but are no longer support	`<center></center>` `<font color="red"></font>`	**12**	**9**
Invalid Construct	Using elements, attributes, or properties that do not exist and never have	`<sidebar></sidebar>`	**12**	**3**
Valid But Unsuitable Construct	Using a familiar but cumbersome element, instead of a simpler and more suitable one	`<p>1. First item</p>` `<p>2. Second item</p>`	**3**	**1**
Misidentified Construct	Using the wrong name to reference a construct	`font-color instead of color` `align instead of text-align`	**24**	**6**
Hyperlink Concepts	Confusing the hyperlink content and destination	`<a href="Google">` `http://google.com</a>`	**7**	**0**
Resource Paths	Errors in constructing the path to a resource such as an image or web page	`http:icer-conference.org` `absolute vs. relative paths`	**1**	**0**

Lists and List Items	Giving a list element a child other than a list item, which is required	`<ol>`
		`<p>Item one</p>`
		`</ol>`
			13	**11**

Ordered List Numbering	Manually numbering ordered list items, which are automatically numbered	`<ol>`
		`<li>1. Item one</li>`
		`<li>2. Item two</li>`
		`</ol>`
			9	**3**

Empty Element Syntax	Errors with empty elements, which are solitary instead of paired like typical elements	`<img src="image.png"></img>` `</ br> instead of  `	**11**	**9**

Style Element Placement	Using style elements outside of head without the scoped attribute	`<body>`
		`<style>`
		`h1 {font-color: red;}`
		`</style>`
			3	**2**

Inline Style Syntax	Syntax errors while writing CSS code inline with HTML	`<h1 color: red;>Header</h1>`	**6**	**1**
Color Hex Values	Misformatting hexadecimal values, which require a hash and 3 or 6 digits	`color: 0000FF;`	**2**	**0**
Missing Units	Forgetting required units on CSS values	`margin: 40;`	**3**	**2**
Naming Identifiers	Starting a class or ID name with a numeral or other invalid character	`<div class="1"></div>`	**3**	**1**

Mistargeted Style	Applying style to wrong element due to a logic error	`table {`
		`text-align: center;`
		`}`
			4	**0**

Overriding Rules	Inadvertently overriding rules due to the CSS cascade	`a:hover {`
		`  color: red;`
		`}`
		
		`a:link {`
		`  color: blue;`
		`}`
			1	**0**

Invisible Elements	Missing content, height, border, or background, causing an element to not be visible as expected	`<div style="width: 500px;"></div>`	**8**	**2**

Centering Block Elements	Inability to center block elements, which requires setting a width, and left and right margins to auto	`<div align="center">Not</div>`
		
		`div {`
		`  text-align: center;`
		`}`
			4	**1**

Collapsing Margins	Undesired collapsing of vertical margins in adjacent or nested elements	`<div style="margin: 10px;">`
		`</div>`
		`<div style="margin: 20px;">`
		`</div>`
			3	**2**

Non-unique IDs	Using an ID multiple times in a document	`<div id="section1">`
		`  <h1 id="section1">1</h1>`
		`</div>`
			1	**0**

Comment Syntax	Syntax errors for comments in HTML and CSS	`// HTML comment` `/ CSS comment`	**4**	**0**
			134	**53**

Table 8. Knowledge-Based Error Types.

Error Type	Description	Examples	Total	Unresolved
HTML Foundations	The basic syntax and semantics of HTML elements, including tags, attributes, and values	`<align="right">Sidebar</align>`	**39**	**17**
CSS Foundations	The basic syntax and semantics of CSS rule sets, including basic selectors, properties, and values	`div: color: red;`	**26**	**12**
CSS Selector	Advanced and compound CSS selectors	`.div > #element`	**23**	**15**
Box Model	Setting the dimensions of elements using properties of the box model	`width, height, padding, border, margin`	**2**	**1**
Positioning Model	Setting the position of an element within the document's flow	`position, float, top, right, bottom, left, display`	**6**	**5**
			96	**50**

5. DISCUSSION

In the following sections, we discuss the implications of this taxonomy of errors in terms of learning basic web development and designing tools for beginners.

5.1 Role of Errors in Learning

The taxonomy presented above helps to map the landscape of errors that people commonly make in HTML and CSS. Although errors are undesirable in many situations, they can play an integral role in teaching and learning. For us, the goal is not to identify errors in order to eliminate them; rather, it is to understand when and why they occur in order to provide learners with the means to detect, understand, and resolve them productively.

At the knowledge-based level, we have identified several topics fundamental to web development. These topics can be roughly ordered by the sophistication of understanding required, with certain topics building on others. The topics suggest different conceptual plateaus on which people are operating. Prior to HTML and CSS foundations, people may have acquired meaning about bits of unconnected code. Upon learning these foundations, they are able to construct the building blocks of web pages, HTML elements and CSS rule sets. Finally, through CSS selectors, they learn how these bits of CSS and HTML can be related to each other in more sophisticated ways, and, through the box and positioning models, they are able to see how all of the parts relate to the whole document.

At the rule-based level, errors give particular insight into the misconceptions people hold about HTML and CSS. At this level, people are applying rules with intention that, while producing errors, make sense according to their current state of knowledge. In many cases, these are rules that have served effectively in the past, but are not workable in exceptional circumstances or changing contexts. Table 7 suggests a number of common misconceptions that students and instructors alike should be vigilant about when introducing topics.

Finally, while skill-based errors were caused by small slips that were usually corrected, they sometimes cascaded into other errors and had the potential to cause surprisingly great difficulties. Skill-based errors, though seemingly minor, often resisted detection and resolution because participants tended to overlook them and misdirect their debugging efforts primarily to less familiar code.

5.2 Implications for Design

Our study gives us insight into how systems can be designed to provide better support for in detecting and fixing web development errors. At all levels, feedback provided by the web editor's live preview panel was instrumental in detecting and resolving errors. As participants typed their code, they were able to immediately test it as the page rendered in real time. However, as the only mode of feedback, the live preview could also be detrimental. Browsers are tolerant of errors, and often render HTML and CSS code that is riddled with bugs. When a beginner writes code that has many errors but still renders as desired, they receive positive feedback in the form of the properly formatted web page. These errors are latent, remain unresolved, and reinforce faulty understandings that can become difficult to overcome.

HTML and CSS validators can detect syntax errors in the code and help counter false reinforcement, whether as an integrated feature of an editor or as a step in beginners' workflow. Beyond syntax errors, HTML and CSS linters apply heuristics that identify common semantic errors that a validator might not catch. For instance, Ko and Wobbrock [8] describe the uniqueness heuristic, which states that an identifier, such as an HTML ID or class, that is used only once is likely unintended. Our taxonomy suggests a number of additional warning signs for semantic errors, particularly at the rule-based level. A div that is being given visual styles but that is not visible due to being dimensionless is one such example.

Finally, errors at each level are best addressed by different approaches, due to differences in their intentionality and the extent of faulty knowledge at their root. Skill-based errors are unintentional, requiring only an indication of their existence and location. Rule-based errors require relatively simple explanations of the errors. At the knowledge-based level, a flood of error messages may be counter-productive, and users may be best served by being directed to substantive learning resources. In short, the skills-rules-knowledge framework suggests how error feedback might be triaged to be most effective.

5.3 Limitations

In this study, we observed the coding behavior of participants directly. This gave us a richer view of coding activity than would have been possible through code inspection or interviews. Coding was accompanied with verbal articulations, facial expressions, gaze changes, web searches, and even different postures, all of which helped us when interpreting and classifying their errors. However, there were significant tradeoffs with this approach. Our analysis was time consuming, which limited the number of participants and the diversity of the coding activities we could observe. In future work, we hope to complement this study by analyzing a broader corpus of data.

6. CONCLUSION

In this paper, we have reported on the errors people make while writing HTML and CSS code. We have found the skills-rules-knowledge framework to be a valuable analytic tool, constructed a taxonomy of errors, and examined the source of errors. We continue to iterate on the openHTML browser based on these findings.

This work is supported by the NSF, CNS #1152094.

7. REFERENCES

[1] Anderson, J. & Jeffries, R. (1985). Novice LISP errors: Undetected losses of information from working memory. *Human-Computer Interaction*, *1*(2), 107–131.

[2] Blackwell, A. (2002). First steps in programming: A rationale for attention investment models. *HCC*, 2–10.

[3] Brandt, J., Guo, P., Lewenstein, J., Dontcheva, M., & Klemmer, S. (2009). Two studies of opportunistic programming: Interleaving web foraging, learning, and writing code. *CHI*. 1589-1598.

[4] Bruckman, A. & Edwards, E. (1999). Should we leverage natural-language knowledge?: An analysis of user errors in a natural-language-stye programming language. *CHI*, 207-214.

[5] Désilets, A., Paquet, S., & Vinson, N. (2005). Are wikis usable? *WikiSym*, 3-15.

[6] Dorn, B. & Guzdial, M. (2010). Learning on the job: Characterizing the programming knowledge and learning strategies of web designers. *CHI*, 703-712.

[7] Ko, A. & Myers, B. (2005). A framework and methodology for studying the causes of software errors in programming systems. *Journal of Visual Languages and Computing*, *16*, 41–84.

[8] Ko, A. & Wobbrock, J. (2010). Cleanroom: Edit-time error detection with the uniqueness heuristic (pp. 7–14). *VL/HCC*, 7-14.

[9] Miller, C., Perkovic, L., & Settle, A. (2010). File references, trees, and computational thinking. *ITiCSE*, 132-136.

[10] Panko, R. (1998). What we know about spreadsheet errors. *Journal of End User Computing*, *10*(2), 15–21.

[11] Park, T., Saxena, A., Jagannath, S., Wiedenbeck, S., & Forte, A. (2013). openHTML: Designing a transitional web editor for novices. *CHI Extended Abstracts*.

[12] Park, T. & Wiedenbeck, S. (2011). Learning web development: Challenges at an earlier stage of computing education. *ICER*, 125-132.

[13] Rasmussen, J. (1983). Skills, rules, and knowledge; Signals, signs, and symbols, and other distinctions in human performance models. *IEEE Transactions on Systems, Man, and Cybernetics*, *13*(3), 257–266.

[14] Reason, J. (1990). *Human Error*. Cambridge University Press.

[15] Rosson, M., Ballin, J., & Nash, H. (2004). Everyday programming: Challenges and opportunities for informal web development. *VL/HCC*, 123-130.

[16] Spohrer, J. & Soloway, E. (1986). Alternatives to construct-based programming misconceptions. *CHI*, 183-191.

[17] Strauss, A. and Corbin, J. (1998). *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. Sage Publications.

[18] Youngs, E. (1974). Human errors in programming. *International Journal of Man-Machine Studies*, *6*, 361–376.