# An Analysis of HTML and CSS Syntax Errors in a Web Development Course

THOMAS H. PARK, Drexel University
BRIAN DORN, University of Nebraska at Omaha
ANDREA FORTE, Drexel University

Many people are first exposed to code through web development, yet little is known about the barriers beginners face in these formative experiences. In this article, we describe a study of undergraduate students enrolled in an introductory web development course taken by both computing majors and general education students. Using data collected during the initial weeks of the course, we investigate the nature of the syntax errors they make when learning HTML and CSS, and how they resolve them. This is accomplished through the deployment of openHTML, a lightweight web-based code editor that logs user activity. Our analysis reveals that nearly all students made syntax errors that remained unresolved in their assessments, and that these errors continued weeks into the course. Approximately 20% of these errors related to the relatively complex system of rules that dictates *when* it is valid for HTML elements to be nested in one another. On the other hand, 35% of errors related to the relatively simple tag syntax determining *how* HTML elements are nested. We also find that validation played a key role in resolving errors: While the majority of unresolved errors were present in untested code, nearly all of the errors that were detected through validation were eventually corrected. We conclude with a discussion of our findings and their implications for computing education.

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

General Terms: Human Factors

Additional Key Words and Phrases: Web development, code editors, computational literacy

**ACM Reference Format:**
Thomas H. Park, Brian Dorn, and Andrea Forte. 2015. An analysis of HTML and CSS syntax errors in a web development course. ACM Trans. Comput. Educ. 15, 1, Article 4 (March 2015), 21 pages.
DOI: http://dx.doi.org/10.1145/2700514

## 1. INTRODUCTION

Web development courses are offered across numerous disciplines, with varying structures that reflect different curricular roles. When web development appears in computer science courses, it is typically used as an engaging context for learning about programming. In these courses, emphasis is placed on using JavaScript, the de facto scripting language of the web, as the basis for learning the fundamentals of programming. In disciplines such as media design, communications, business, and the arts, web development courses put more focus on the practical skills needed to build and maintain

This work is supported by the National Science Foundation, under grant CNS-1152094.
Authors' addresses: T. H. Park and A. Forte, College of Computing and Informatics, Drexel University; emails: {park, aforte}@drexel.edu; B. Dorn, Department of Computer Science, University of Nebraska at Omaha; email: bdorn@unomaha.edu.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2015 ACM 1946-6226/2015/03-ART4 $15.00
DOI: http://dx.doi.org/10.1145/2700514

web pages rather than fundamental concepts of programming. Introductory courses in these disciplines tend to devote more time to auxiliary topics like web servers, multimedia, and information architecture. These courses customarily spend most of their time with HTML and CSS, teaching just enough JavaScript to learn the role it plays on the web and implement basic interactive effects. While courses of the latter type involve less programming, they still offer students the opportunity to learn a great deal about computing. Despite this potential, it remains unclear how to design these courses to make the most of what is, for many students, an interest-driven, formative experience with computing. A first step is developing a better understanding of the experiences beginners have when learning HTML and CSS.

In this study, we investigate the difficulties students face when learning HTML and CSS in an introductory web development course. Specifically, we examine the types of syntax errors students commonly make, how they are able to resolve them, and the role validation plays in these outcomes. We accomplish this by analyzing coding activity logged by an instrumented code editor called openHTML. Through the use of openHTML, we are able to collect fine-grained data on student activity that is not typically available to course instructors.

Our study focuses on syntax errors because they can present a significant hurdle for first-time coders. Like the programming languages taught in more traditional CS courses, markup languages like HTML and style sheet languages like CSS require students to follow precise rules of syntax. A missing semicolon or quotation mark can be the difference in whether the web page is functional or not. The challenges of obeying syntax rules are compounded when they must be juggled with higher-level concerns about semantics and design.

Our work makes two main contributions. First, we characterize the difficulties students face learning HTML and CSS through the analysis of syntax errors. To our knowledge, our study is the first to offer a fine-grained analysis of student coding activity in HTML and CSS as they progress in a web development course. Our results shed light on the types of errors students make in HTML and CSS, how they change over time, how successful students are in resolving them, and the role validation plays in these outcomes. Second, we report on our use of openHTML, an experimental web editor, as a pedagogical tool and research instrument.

In the next section, we review relevant research from the computing education literature. In Section 3, we provide a description of our study design, our experimental web editor called openHTML, and the web development course. In Section 4, we present our analysis of syntax errors made during the course. Finally, in Section 5, we discuss the implications of our findings for computing education.

## 2. RELATED WORK

Consistent with a constructionist approach to learning that promotes making personally meaningful and publicly accessible artifacts [Papert and Harel 1991], researchers have noted that students, particularly nonmajors, are motivated by producing programs that are relevant to their interests, as opposed to building computer programs from scratch using general-purpose programming languages [Mercuri et al. 1998]. Numerous case studies demonstrate how web development has been leveraged for this purpose [Lim 1998; Walker and Browne 1999; Klassner 2000; Reed 2001; Treu 2002; Sridharan 2004; Wang and Zahadat 2009]. Studies of web development practitioners, including both professionals and hobbyists, also illustrate how people from diverse backgrounds are attracted to computing through the Web [Rosson et al. 2004, 2005; Dorn and Guzdial 2010b].

Web development is a rich activity that encompasses many topics including visual design, information architecture, and programming. While this richness is one of the very qualities that give it broad appeal, it has created many challenges for teachers and curriculum makers, which they have addressed by balancing breadth and depth [Reed 2001; Treu 2002; Sridharan 2004], synthesizing multiple topics into a comprehensive project, and proposing multicourse sequences [Connolly 2011]. Although educators have recognized the diverse topics involved in web development, research has focused primarily on its role in learning to program. For instance, Dorn and Guzdial [2010a] used a card-sorting task to examine the knowledge professional web developers have of programming concepts such as assignment, loops, and recursion.

Surprisingly, HTML and CSS, two fundamental languages of the web, have been the subject of little research to date. As with conventional programming languages like JavaScript, HTML and CSS present some of the same difficulties through “syntax errors, runtime errors, or bugs in the form of unintended or exceptional behaviors” [Blackwell 2002]. They also invoke many of the concepts, practices, and skills central to learning about computation [Brennan and Resnick 2012]. Two studies highlight the potential for exploring this nonprogramming terrain: A study of elementary students creating wikis investigates the difficulties students have with hyperlinks [Désilets et al. 2005], while a second study of university students in a web development course analyzes their understanding of tree structures and file reference paths [Miller et al. 2010].

Students have previously been found to encounter many barriers related to HTML and CSS in an introductory web development course [Park and Wiedenbeck 2011]. A follow-up lab study in which participants were tasked with constructing simple web pages revealed that approximately 70% of the errors they made involved invalid syntax, supporting the notion that even the syntax of HTML and CSS can act as obstacles to successfully authoring web pages [Park et al. 2013b]. Our study builds on this work by shifting from single sessions in a lab to students in a live course over the span of several weeks. This change allows us to analyze students’ coding behavior as they progress through an undergraduate course, thereby increasing the ecological validity of our findings.

While the literature on learning HTML and CSS is limited, our study is informed by the extensive research on novice programming that recognizes the challenges syntax creates for beginners learning to program [Du Boulay 1986; Spohrer and Soloway 1986; Pea et al. 1987]. For instance, Stefik and Siebert [2013] conducted an empirical study comparing the accuracy of novices writing simple programs using a variety of programming languages. They reported that differences in syntax among languages like Perl, Python, and Ruby have a significant impact on the perceived and actual difficulty faced by beginners. This leads us to consider how beginners are affected by the syntax of HTML and CSS, which represent entirely different classes of computing languages.

Our work is also informed by a study that used an instrumented programming environment to catalog syntax errors and characterize the compilation behavior of Java students [Jadud 2006]. Jadud proposes Error Quotient (EQ) as a measure of a student’s struggles with syntax within a programming session, based on frequency, type, and location of syntax errors in consecutive compilation attempts. Watson et al. [2014] recently compared such measures of programming behavior and found that compared to traditional test-based measures, they were significantly better predictors of student performance in an introductory programming course.

## 3. STUDY

In order to develop a better understanding of the experiences beginners have learning HTML and CSS, our study was guided by the following research questions:

Table I. Weekly Overview of the Course Schedule, Including Key Topics and Assessments

| Week | Topics | Assessments |
|---|---|---|
| 1 | Internet and web basics | |
| 2 | Structural basics | Lab 0 |
| 3 | Structural basics, links | Lab 1, Lab 2 |
| 4 | Introduction to CSS, visual elements, and graphics | Lab 3, Assignment 1 |
| 5 | Wireframing, mockups, web design best practices | Assignment 2 |
| 6 | More CSS, page layouts | Assignment 3 |
| 7 | Page layouts, uploading to servers | Assignment 4<br>Begin Project 1 |
| 8 | Midterm exam | |
| 9 | User-centered design, usability testing | Project 1 due |
| 10 | HTML5 structural elements, tables | Begin Project 2 |
| 11 | Designing navigation, site maps, forms | |
| 12 | Media, interactivity, and advanced selectors | Assignment 5 |
| 13 | JavaScript basics, jQuery | |
| 14 | Accessibility evaluation | |
| 15 | Publishing, hosting, and search engine optimization | |
| 16 | Final exam, presentations | Project 2 due |

RQ1: What types of syntax errors do students in an introductory web development course make when learning HTML and CSS?
RQ2: How well are students able to resolve different types of HTML and CSS syntax errors, and what role does validation play in their resolution?

To address these questions, we relied on online logs generated through the use our experimental web editor called openHTML. We conducted the study for two semesters of a course taught by the second author during Fall 2012 and Spring 2013.

### 3.1. Course

The course studied here introduces undergraduate students to the fundamentals of front-end web development. By the end of the course, students are expected to be able to design and implement basic websites using HTML, CSS, and small amounts of JavaScript. Students are also taught to follow a systematic, user-centered design process and author code that complies with web standards. Both general education students and CS majors can take the course, either as a standalone or as part of an undergraduate major or minor in web design and development. The course typically draws a range of noncomputing students majoring in graphic design, audio engineering, and the humanities.

Assessments of varying scope were used in the class. Labs consisted of small coding tasks started in class. Assignments were midsized, closed-form homework tasks based on end-of-chapter projects in the Felke-Morris [2011] text (e.g., creating web pages with fully fleshed-out style and content). Two larger, open-ended projects, a personal web portfolio and site redesign, consisted of multiple components and each spanned several weeks. Lastly, midterm and final exams were administered. Although minor adjustments were made between the two semesters, Table I shows a representative schedule with topics and assessments organized by week.

An explicit requirement for Assignments 1 and 2 was that students tested their code and ensured that it passed HTML and CSS validation. The World Wide Web Consortium [2014], the governing standards organization of the web, outlines the benefits of validation, including that it

—teaches good practices for beginners by helping them spot mistakes and introducing broader quality concepts such as accessibility;

Table II. Description of Assessments Using openHTML
Labs are small, one-off in-class activities. Assignments are moderate-sized homework assessments that build on previous activities.

| Assessment | Description |
|---|---|
| Lab 0 | Create a web page with information about what you did during the last break, using basic HTML tags such as title, header, paragraph, and blockquote. |
| Lab 1 | Create a to-do list using an ordered or unordered list as well as other basic HTML tags. |
| Lab 2 | Copy the web page created in Lab 1 and add a section with your favorite links. |
| Lab 3 | Copy a previously created web page and add an image. Also use CSS to position the image and add a background image to the web page. |
| Assignment 1 | Create a resort homepage with a site logo, navigation menu, content area, and footer using HTML. |
| Assignment 2 | Copy the resort homepage from Assignment 1 to create an index and a subpage. Link the two pages together. Use CSS to style the text and background colors. |

—guards against errors that may not be handled consistently or gracefully across current and future platforms; and
—signals quality and whether the code is clean and well formed, or quickly hacked together.

For this study, assessments from the initial weeks of the course were modified to use openHTML, our experimental web editor. We selected labs and homework assignments from the first five weeks of the course, which involved the creation of individual web pages rather than multipage sites, and did not require the use of JavaScript (see Table II). The remaining assessments, which were not included in this study, were completed with Aptana Studio, a full-featured integrated development environment based on Eclipse.

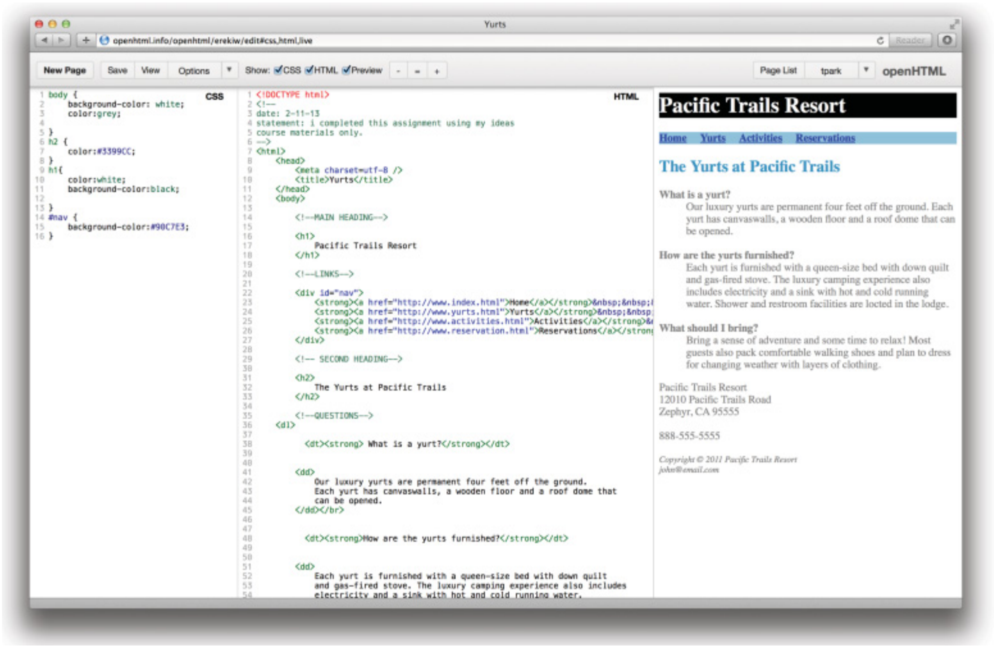
### 3.2. openHTML Editor

We have been developing the openHTML editor to support people who are learning to code as well as the educators who teach them. The goal of openHTML is to make it easy for beginners to author simple web pages by writing code in HTML and CSS. Our approach is to make code the focus of the interface, abstracting away many of the noncoding concerns. This contrasts with the direct manipulation approach taken by WYSIWYG editors, which fundamentally changes the nature of web development by shielding users from the code.

We have taken a design-based research approach [Barab and Squire 2004], iteratively developing new features, deploying them in the classroom and other natural settings (e.g., Park et al. [2013a]), and evaluating their impact. The two semesters in this study coincide with two rounds of this iterative design process.

We were inspired by a previous study finding that nearly 25% of help-seeking episodes in a web development course were related to difficulties students had installing and configuring their development environments. These technological issues caused frustration among students and distracted them from what they considered to be their primary learning goals [Park and Wiedenbeck 2011].

Because openHTML is implemented in the browser, it does not require installation: Both the tool itself and user data can be accessed from any computer with an Internet connection. The editor view has a minimal interface comprised of three panes: one for HTML input, another for CSS input, and a third that renders a preview of the web page that updates as users modify the code (Figure 1). Users can undo their code edits and revert to previously saved revisions, which along with the instant feedback aims at reducing any risks associated with experimentation.


Yurts
openhtml.info/openhtml/erekiw/edit#css,html,live
Reader
New Page
Save
View
Options
Show: ✓CSS ✓HTML ✓Preview
Page List
tpark
openHTML
CSS
HTML

```
body {
    background-color: white;
    color:grey;

}
h2 {
    color:#3399CC;
}
h1{
    color:white;
    background-color:black;

}
#nav {
    background-color:#90C7E3;
}
```

```
<!DOCTYPE html>
<!--
date: 2-11-13
statement: i completed this assignment using my ideas
course materials only.
-->
<html>
    <head>
        <meta charset=utf-8 />
        <title>Yurts</title>
    </head>
    <body>

        <!--MAIN HEADING-->

        <h1>
            Pacific Trails Resort
        </h1>

        <!--LINKS-->

        <div id="nav">
            <strong><a href="http://www.index.html">Home</a></strong>&nbsp;&nbsp;
            <strong><a href="http://www.yurts.html">Yurts</a></strong>&nbsp;&nbsp;
            <strong><a href="http://www.activities.html">Activities</a></strong>&
            <strong><a href="http://www.reservation.html">Reservations</a></strong>
        </div>

        <!-- SECOND HEADING-->

        <h2>
            The Yurts at Pacific Trails
        </h2>

        <!--QUESTIONS-->
    <dl>

          <dt><strong> What is a yurt?</strong></dt>


        <dd>
            Our luxury yurts are permanent four feet off the ground.
            Each yurt has canvaswalls, a wooden floor and a roof dome that
            can be opened.
        </dd></br>


          <dt><strong>How are the yurts furnished?</strong></dt>


        <dd>
            Each yurt is furnished with a queen-size bed with down quilt
            and gas-fired stove. The luxury camping experience also includes
            electricity and a sink with hot and cold running water.
```

Pacific Trails Resort

Home Yurts Activities Reservations

The Yurts at Pacific Trails

What is a yurt?
Our luxury yurts are permanent four feet off the ground. Each yurt has canvaswalls, a wooden floor and a roof dome that can be opened.

How are the yurts furnished?
Each yurt is furnished with a queen-size bed with down quilt and gas-fired stove. The luxury camping experience also includes electricity and a sink with hot and cold running water. Shower and restroom facilities are locted in the lodge.

What should I bring?
Bring a sense of adventure and some time to relax! Most guests also pack comfortable walking shoes and plan to dress for changing weather with layers of clothing.

Pacific Trails Resort
12010 Pacific Trails Road
Zephyr, CA 95555

888-555-5555

*Copyright © 2011 Pacific Trails Resort*
*john@email.com*

Fig. 1. openHTML editor view. The left pane is for inputting CSS, the middle pane for HTML, and the right pane is a live preview of the web page. The toolbar at the top has options such as saving the web page, viewing it in its own window, validating the HTML and CSS, and toggling the panes.

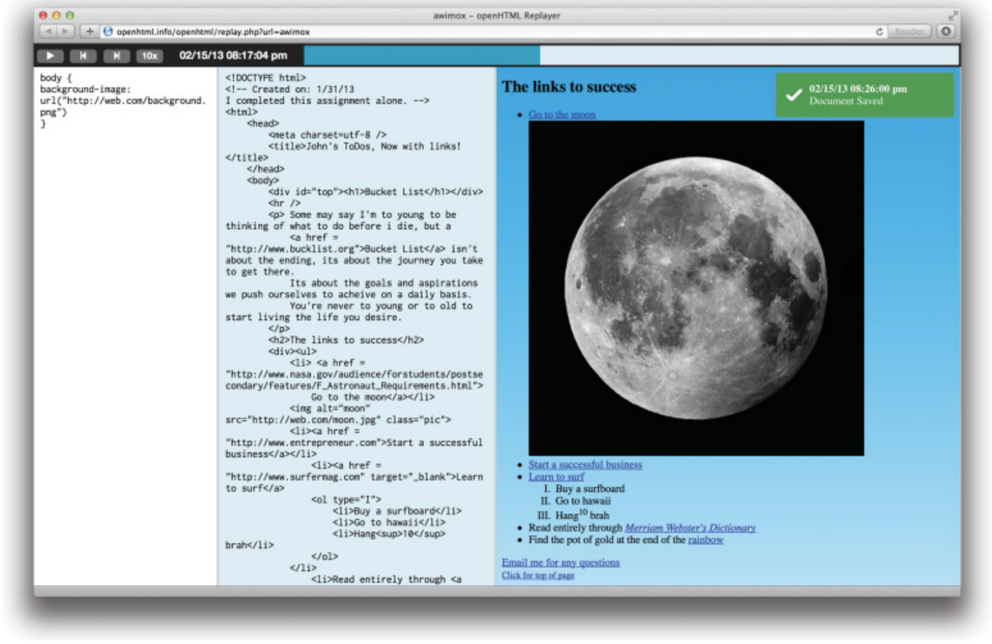
Rendered web pages and the underlying source code are publicly accessible by default, allowing users to share their creations without setting up a web host or uploading files. However, web pages are not discoverable and can only be accessed if their URLs are explicitly shared, granting learners some measure of privacy. Any user can fork a web page and make edits to their copy of it. Use cases for this feature include students creating multiple variations of a page and teachers seeding a web page with code for students to use as a template.

In the second semester, we added HTML and CSS validators for use with Assignments 1 and 2. By making use of these features, students were able to validate their current webpage's HTML or CSS code, opening a new browser tab that listed the syntax errors that were detected in the code. The validators made use of the markup validation service APIs provided by the World Wide Web Consortium. While students in the first term were also required to have their assignments pass validation, this involved visiting an external site, copying the code from openHTML, and pasting it into the external validator. The teacher reported that this was a cumbersome process for students. It also prevented us from collecting data about their validation attempts, which we recognized as a valuable source of insight into how they dealt with syntax errors over time.

We also implemented a data logger in the second semester that records coding activity at the keystroke level and replays coding sessions (Figure 2). In addition to HTML and CSS coding activity, it logged events such as saving and validating web pages.

## 3.3. Procedure

We performed a log analysis of the HTML and CSS coding activity of all of the students in the course. This study builds on our previous work [Park et al. 2013b] where we used


awimox – openHTML Replayer
openhtml.info/openhtml/replay.php?url=awimox
10x
02/15/13 08:17:04 pm
Reader

```css
body {
background-image:
url("http://web.com/background.
png")
}
```

```html
<!DOCTYPE html>
<!-- Created on: 1/31/13
I completed this assignment alone. -->
<html>
    <head>
        <meta charset=utf-8 />
        <title>John's ToDos, Now with links!
</title>
    </head>
    <body>
        <div id="top"><h1>Bucket List</h1></div>
        <hr />
        <p> Some may say I'm to young to be
thinking of what to do before i die, but a
            <a href =
"http://www.bucklist.org">Bucket List</a> isn't
about the ending, its about the journey you take
to get there.
            Its about the goals and aspirations
we push ourselves to acheive on a daily basis.
            You're never to young or to old to
start living the life you desire.
        </p>
        <h2>The links to success</h2>
        <div><ul>
            <li> <a href =
"http://www.nasa.gov/audience/forstudents/postse
condary/features/F_Astronaut_Requirements.html">
                Go to the moon</a></li>
            <img alt="moon"
src="http://web.com/moon.jpg" class="pic">
            <li><a href =
"http://www.entrepreneur.com">Start a successful
business</a></li>
                <li><a href =
"http://www.surfermag.com" target="_blank">Learn
to surf</a>
                <ol type="I">
                    <li>Buy a surfboard</li>
                    <li>Go to hawaii</li>
                    <li>Hang<sup>10</sup>
brah</li>
                </ol>
            </li>
                <li>Read entirely through <a
```

**The links to success**

- Go to the moon
- Start a successful business
- Learn to surf
  I. Buy a surfboard
  II. Go to hawaii
  III. Hang$^{10}$ brah
- Read entirely through *Merriam Webster's Dictionary*
- Find the pot of gold at the end of the rainbow

Email me for any questions
Click for top of page

02/15/13 08:26:00 pm
Document Saved

Fig. 2. openHTML replayer. Coding activity that has been previously recorded is being replayed, including edits to the CSS and HTML, and updates to the rendered web page. The active pane is highlighted, and events such as saving the web page and validating the code are displayed in a pop-up window in the top right corner. The toolbar provides playback controls such as pause, skip, and progress bar.

a think-aloud protocol to determine HTML and CSS errors that were both syntactic and semantic in nature. In this study, our analysis was limited to syntax errors, but we were able to gain access to uninterrupted behaviors in an actual undergraduate course, attaining greater ecological validity in our findings.

We used openHTML to analyze the HTML and CSS syntax errors students made while completing assignments. Typically, only the final submitted version of an assignment is available to instructors; similarly, much of the action that occurs when students learn web development is "researcher distant"—not amenable to direct observation [Fincher et al. 2011]. However, we instrumented openHTML to log each saved revision of each project, providing snapshots of the assignments at varying levels of completion. Because openHTML was used for both in-class labs and homework assignments, this gave us access to the development process of students beyond the classroom. For the second semester of the course, even finer-grained logging was implemented, granting us a keystroke-level view of students' coding activity so that we could replay coding sessions and inspect how HTML and CSS validation was used to detect and resolve errors.

In the first part of the log analysis, which was conducted for both terms of the course, we examined the unresolved syntax errors present in the final version of the students' code. These represent errors that students did not detect, or were unable to resolve even after validation. To identify and analyze syntax errors, we passed student code through the HTML and CSS validators and cataloged the error messages that were generated. In the error messages, mentions of specific elements, attributes, or values were then replaced with placeholders, allowing similar error messages to be classified as the same error type. For example, two error messages reported by the HTML validator were “Element h1 not allowed as child of element span in this context” and “Element dt not allowed as child of element body in this context.” Both were classified as the “Element Y not allowed as child of element X in this context” error type.

Following this process, we examined the error types along two dimensions:

—*Frequency*: What are the most common errors in the course? This was operationalized as the total count of each error type.
—*Prevalence*: Which errors affected students most broadly in the course? This was operationalized as the proportion of unique students making a type of error at least once.

We also analyzed the errors by tallying mentions of language constructs in the error messages in order to uncover the elements, properties, and other constructs that were most problematic for students. This provided context for the circumstances in which the errors were made.

In the second part of log analysis, we examined snapshots of the students’ code each time they validated their code. The assumption is that as a result of validation, students become aware of bugs present in their code and make an effort to resolve them. This analysis therefore reveals all of the errors students encountered, rather than only the ones remaining in the final submissions. It also allows us to determine the degree to which errors recurred in multiple validations, whether students were having difficulty resolving the same errors or were repeating similar ones. Along with frequency and prevalence, we analyzed the errors along two additional dimensions:

—*Recurrence*: How deeply each did type of error affect students? This was operationalized as the median number of consecutive validations for which an error persisted.
—*Resolution*: How successful were students in resolving each type of error? This was operationalized by comparing the errors found during validation with the ones that were still present in the final submissions.

The second part of the analysis was limited to Assignments 1 and 2 in the Spring 2013 term when the validator feature was added to openHTML.

### 3.4. Participants

The log analysis included the activity of 23 students in the course (9 in Fall 2013 and 14 in Spring 2013). 12 of these students (4 female, 8 male) agreed to interviews about their experiences with web development and programming prior to the course, which has been shown to predict the success of nonmajors in learning to program [Wiedenbeck 2005]. Interviews were conducted in the first week of the course for the Spring 2013 term and near the end of the course for the Fall 2012 term due to scheduling constraints.

The interview participants averaged 21 years of age and ranged from the first to fourth year of their university program. Students were pursuing a variety of majors, including web design and development, computer science, audio engineering, communications, and business. The course was a requirement for some and elective for others. Two of the participants, P9 and P11, withdrew partway through the course. Data they submitted before they withdrew from the course was included in the analysis. Demographic data are provided in Table III.

We were surprised to find that all of the participants we interviewed had experience with HTML and CSS before the course. Students reported their level of prior experience with HTML, CSS, and JavaScript as none (0), beginner (1), intermediate (2), or expert (3). All participants indicated that they had at least beginner experience in HTML, with an average rating of 1.50 (SD = 0.52). CSS and JavaScript were less familiar, with a mean of 1.17 (SD = 0.39) and 0.75 (SD = 0.75), respectively.

Table III. Interview Participants' Age, Gender, Major, Term, and Self-Reported Experience with HTML, CSS, and JavaScript Prior to the Course
Prior experience with HTML, CSS, and JavaScript are given on a scale of 0 to 3. The self-efficacy questionnaire was not administered in Fall 2012.

| Code | Age | Gender | Major | Term | HTML | CSS | JS |
|---|---|---|---|---|---|---|---|
| P1 | 20 | Female | Graphic Design | Fall 2012 | ● | ● | |
| P2 | 20 | Female | Web Design and Development | Fall 2012 | ● | ● | |
| P3 | 19 | Female | Web Design and Development | Fall 2012 | ●● | ●● | |
| P4 | 19 | Male | Computer Science | Spring 2013 | ● | ● | |
| P5 | 19 | Male | Audio Engineering | Spring 2013 | ●● | ● | |
| P6 | 21 | Male | Audio Engineering | Spring 2013 | ● | ● | ● |
| P7 | 21 | Female | Biology | Spring 2013 | ●● | ● | ●● |
| P8 | 24 | Male | Computer Science | Spring 2013 | ● | ● | ● |
| P9 | 19 | Male | Audio Engineering | Spring 2013 | ●● | ● | ●● |
| P10 | 21 | Male | Communications | Spring 2013 | ●● | ●● | ● |
| P11 | 28 | Male | Communications | Spring 2013 | ● | ● | ● |
| P12 | 19 | Male | Business | Spring 2013 | ●● | ● | ● |

These earlier experiences tended to be limited, and students rarely recalled more than a few basic HTML elements from them. Nevertheless, they expressed that these experiences were beneficial, allowing them to relate new information to earlier knowledge.

> "...I was dealing with code. I didn't realize that's what I was doing then, but now, like when we learned a couple of HTML things, I was like oh, I knew that... It came in handy. I guess just like giving me confidence with things, like we would learn something and it wasn't totally foreign. I'd just kind of know what [the teacher] was talking about." (P1)

Only two students reported taking another web development course before this course. Instead, students were primarily exposed to HTML, CSS, and JavaScript through informal activities on popular web services.

> "I think definitely the first time I ever used HTML was just for like having MySpace and I wanted a little bit more creative control, so I just kind of like learned. I knew a basic set of elements." (P6)

Several students reported learning basic HTML tags and CSS styles in order to customize profiles on social networking sites like MySpace, modify templates on blogging platforms like Tumblr and WordPress, and create attention-grabbing posts on the classified advertising site Craigslist.

In their prior experiences, students generally took a more opportunistic approach to web development [Brandt et al. 2009], engaging in just-in-time learning to tweak existing code and personalize their content. They relied on web searches to find relevant information and snippets of code, which they often reused without a thorough understanding through trial and error:

> "I looked up things on different web browsers and it was kind of very much like copy and paste code work. I didn't really understand what I was doing, but I understood that I could use those things and just change the values to whatever variables I needed." (P6)

Students indicated that they were often able to accomplish their immediate goals, but did not learn fundamental concepts that enabled them to develop a deeper understanding of web development. Instead, they were on the path to developing "pockets of expertise," like many informal and even professional web developers [Rosson et al. 2004; Dorn and Guzdial 2010a].

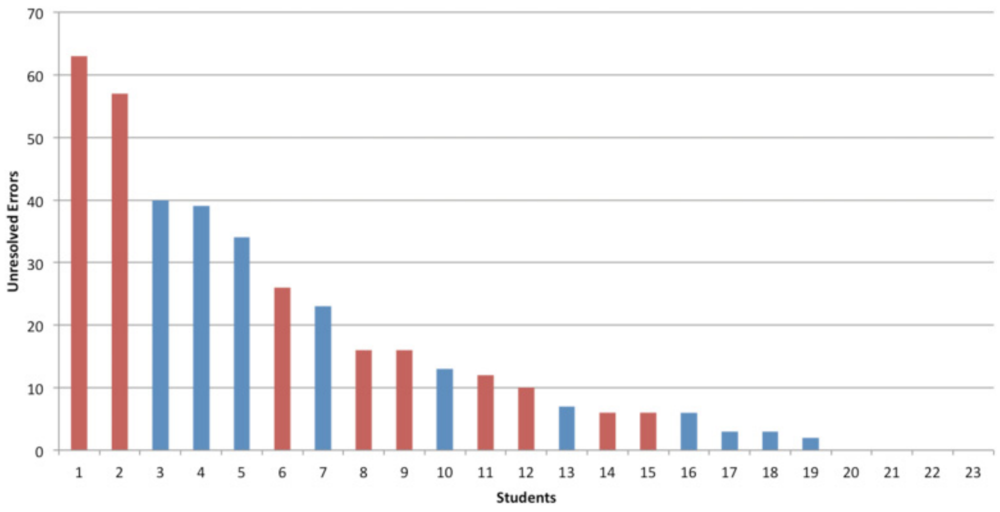


Fig. 3. The number of unresolved errors by student. Students from Fall 2012 are in red, while students from Spring 2013 are in blue. All four of the students without any unresolved errors were from the Spring 2013 semester.

Table IV. Mean Lines of Code, Revisions, and Unresolved errors by Assessment
Standard deviations are given in parentheses.

| Assessment | Lines of Code | Revisions | Unresolved Errors |
|---|---|---|---|
| Lab 0 | 34.6 (12.0) | 17.9 (11.3) | 1.4 (2.3) |
| Lab 1 | 43.1 (9.9) | 21.4 (13.0) | 1.4 (2.8) |
| Lab 2 | 60.6 (20.9) | 17.5 (13.7) | 2.3 (3.2) |
| Lab 3 | 82.9 (44.4) | 20.6 (18.6) | 2.8 (3.7) |
| Assignment 1 | 56.7 (12.0) | 15.3 (12.1) | 3.0 (5.1) |
| Assignment 2 | 166.0 (44.4) | 37.5 (18.0) | 5.9 (6.4) |

## 4. RESULTS

### 4.1. Unresolved Errors

In this section, we report on the HTML and CSS syntax errors students were unable to resolve in their assessments. We start with an overview of the errors, and then discuss their relation to two concepts—nesting and parent-child rules, and how they changed as students progressed in the course.

*4.1.1. Overview.* Of the 23 students, 19 had unresolved errors in their lab and assignment submissions, averaging 16.6 errors (SD = 18.4) across the six assessments, up to a maximum of 63. Figure 3 shows the distribution of unresolved errors among students.

Most of these related to HTML (97.4%). This is likely a product of the topics covered in the early part of the course when we collected data, rather than a generalizable proportion of HTML and CSS errors in an entire introductory course.

Table IV describes the six assessments in terms of the average lines of code, revisions, and unresolved errors. Lines of code and revisions are substantially greater for Assignment 2 than the rest of the assessments in part because it required the creation of two distinct web pages as opposed to a single one. In general, as the scope of the assessments grew, so too did the number of revisions students made and the unresolved errors that were present in them.

Table V. Error Types Comprising Unresolved Errors by Frequency and Prevalence

| Error Type | Frequency | Prevalence |
|---|---|---|
| Element Y not allowed as child of element X in this context | 77 | 15 (65%) |
| Unclosed element X | 47 | 10 (43%) |
| End tag X | 48 | 5 (22%) |
| Named character reference was not terminated by a semicolon (or & should have been escaped as &.) | 32 | 6 (26%) |
| No X element in scope but a X end tag seen | 25 | 5 (22%) |
| End tag X seen, but there were open elements | 24 | 6 (26%) |
| Consecutive hyphens did not terminate a comment. – is not permitted inside a comment, but e.g. - - is. | 24 | 1 (4%) |
| End tag for X seen, but there were unclosed elements | 19 | 9 (39%) |
| Stray end tag X | 11 | 8 (35%) |
| Attribute Y not allowed on element X at this point | 9 | 3 (13%) |
| Bad value Z for attribute Y on element X | 8 | 7 (30%) |
| A X start tag seen but an element of the same type was already open | 8 | 6 (26%) |
| Value error | 5 | 3 (13%) |
| End tag X violates nesting rules | 5 | 3 (13%) |
| Saw < when expecting an attribute name. Probable cause: missing > immediately before | 4 | 4 (17%) |
| Element X is missing a required instance of child element Y. | 4 | 3 (13%) |
| Quote " in attribute name. Probable cause: matching quote missing somewhere earlier | 3 | 2 (9%) |
| Parse error | 3 | 2 (9%) |
| A slash was not immediately followed by >. | 3 | 3 (13%) |
| The Y attribute on the X element is obsolete. | 2 | 1 (4%) |
| Stray start tag X | 2 | 1 (4%) |
| Element X is missing one or more of the following attributes: content, itemprop, property | 2 | 1 (4%) |
| Document type does not allow element X here | 2 | 1 (4%) |
| Y is not an X value. | 1 | 1 (4%) |
| There is no attribute X. | 1 | 1 (4%) |
| The X element is obsolete. | 1 | 1 (4%) |
| Text not allowed in element X in this context | 1 | 1 (4%) |
| Self-closing syntax (/>) used on a nonvoid HTML element. Ignoring the slash and treating as a start tag | 1 | 1 (4%) |
| Required attribute X not specified. | 1 | 1 (4%) |
| Property X does not exist. | 1 | 1 (4%) |
| No space between attributes | 1 | 1 (4%) |
| No document type declaration | 1 | 1 (4%) |
| End tag for element X which is not open | 1 | 1 (4%) |
| Element X must not be empty. | 1 | 1 (4%) |
| Duplicate attribute X | 1 | 1 (4%) |
| Bad character - after <. Probable cause: unescaped <. Try escaping it as <. | 1 | 1 (4%) |
| An X element must have a Y attribute, except under certain conditions. | 1 | 1 (4%) |
| –! found in comment. | 1 | 1 (4%) |

The syntax errors were distilled into 38 error types. Error types are shown in Table V, along with their frequency (overall count) and prevalence (proportion of students who made this type of error at least once).

The top ten error types accounted for 81% of the instances, with the remaining types committed by only a small proportion of students. In the following sections, we organize these error types around two key concepts: nesting tags and parent-child rules.

Table VI. The Number of Nesting Errors and the Proportion of Overall Errors Related to Nesting by Assessment

| Assessment | Count | Proportion |
|---|---|---|
| Lab 0 | 11 | 33.3% |
| Lab 1 | 8 | 24.2% |
| Lab 2 | 16 | 30.8% |
| Lab 3 | 14 | 21.5% |
| Assignment 1 | 8 | 12.5% |
| Assignment 2 | 30 | 22.2% |

*4.1.2. Nesting Tags.* Nesting is the organization of elements into multiple levels hierarchy and is a fundamental property of HTML. Regardless of the elements involved, the syntax determining *how* tags are nested follows this consistent pattern: `<parent><child>content</child></parent>`. Eight of the most common errors related directly to managing HTML start and end tags at multiple levels of nesting, comprising 35.1% of the total errors found in the students' final submissions. These included unclosed elements (i.e., missing end tags):

—Stray start tag X
—End tag for X seen, but there were unclosed elements
—End tag X seen, but there were open elements
—Unclosed element X

Extraneous end tags:

—Stray end tag X
—End tag for element X, which is not open
—No X element in scope but an X end tag seen

And errors caused by overlapped nesting (i.e., closing the outer element before the inner element is closed):

—End tag X violates nesting rules

"End tag X" errors were not counted because we determined that they resulted from void elements such as line breaks (`br`) with malformed syntax rather than mistakes related to nesting tags.

Table VI gives the number and proportion of errors related to nesting by assessment. The proportion of errors is a useful point of comparison given differences in the scope of each assessment. The table reveals that nesting errors remained relatively consistent from one assignment to the next.

The HTML constructs reported in the original error messages (Table VII) shed light on when and why beginners are likely to make nesting errors. Nesting error messages occurred most often when dealing with `div` elements. There are several reasons this might be the case. First, `div` elements are simply a frequently used element. Second, they are commonly used at multiple levels of nesting as a generic element to organize content and define page layouts, resulting in `div` elements nested within other `div` elements. Using identical elements in this manner makes tracking different levels of nesting difficult and increases the likelihood of errors, although this can be mitigated through coding practices like indentation.

Inline elements such as `strong`, `em`, `i`, and `small`, were also frequently involved in nesting errors. Beginners who are not yet comfortable with CSS may rely on these HTML elements to bold, italicize, or change the size of text. In order to apply multiple styles, a string of text is wrapped with several tags simultaneously, making them prone

Table VII. A Count of the HTML Elements Mentioned in Error Messages Related to Nesting

| Construct | Count |
|---|---|
| div | 18 |
| body | 17 |
| strong | 15 |
| p | 14 |
| li | 10 |
| small | 9 |
| head | 9 |
| sub | 7 |
| sup | 4 |
| em | 4 |
| dt | 4 |
| ol | 3 |
| i | 3 |
| cite | 3 |
| ul | 2 |
| span | 2 |
| nav | 2 |
| html | 2 |
| title | 1 |
| hr | 1 |
| h1 | 1 |
| a | 1 |

to nesting errors. An aggravating factor is that when multiple inline elements are used, their tags are often written on a single line of code, without the aid of indentation.

Finally, upon closer inspection of the responsible code, errors involving `head` ("Stray end tag head"), `body` ("End tag for body seen, but there were unclosed elements"), and `dt` ("No dt element in scope but a dt end tag seen.") were typically not the result of improper nesting. In HTML, when rules requiring elements to have specific parent or child elements were violated (e.g., placing elements outside of the `head` or `body`), they are implicitly closed or new elements created, resulting in unmatched start or end tags. Thus, web pages with invalid syntax can still often be rendered, making HTML a more forgiving language, but often leading to unexpected behaviors and baffling error messages that hinder debugging. In the next section, we discuss errors that relate to these parent-child rules in detail.

*4.1.3. Parent-Child Rules.* Nesting HTML elements naturally gives rise to a hierarchical structure in the code, where elements contain child elements and are themselves contained by parent elements. These elements have parent-child rules that constrain *when* they can be nested in one another. One example of this is the `html` element, which must be at the root level and can only contain one `head` followed by one `body` element.

Most other elements have more freedom in where they can be nested but are also governed by such rules. However, our analysis suggests that these rules were frequently unfamiliar to or not well understood by the students. Three of the error types related to rules that dictate when elements can be nested, accounting for 21.5% of all unresolved syntax errors:

—Element Y not allowed as child of element X in this context
—Text not allowed in element X in this context
—Element X is missing a required instance of child element Y

Table VIII. The Number of Parent-Child Errors and the Proportion of Overall Errors Related to Parent-Child Relationships by Assignment

| Assignment | Count | Proportion |
|---|---|---|
| Lab 0 | 3 | 9.1% |
| Lab 1 | 10 | 30.3% |
| Lab 2 | 12 | 23.1% |
| Lab 3 | 13 | 20.0% |
| Assignment 1 | 3 | 4.7% |
| Assignment 2 | 41 | 30.4% |

Table IX. The Most Common HTML Elements Mentioned in Error Messages Related to Parent-Child Rules

| | | Parent | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | **body** | **head** | **ol** | **ul** | **dl** | **div** | **strong** | **small** |
| | **dt** | 10 | | | | | 6 | | |
| | **dd** | 10 | | | | | 6 | | |
| | **title** | 2 | 6 | | | | | | |
| | **ul** | | | 7 | 4 | | | | |
| **Child** | **blockquote** | | | | | | | | |
| | **hr** | | | | | | | 3 | 2 |
| | **br** | | | | 4 | 6 | | | |
| | **a** | | | | 3 | | | | |

The proportion of errors related to parent-child rules had more variation than did nesting errors from one assignment to the next. As seen in Table VIII, they were most common in Lab 1, Lab 2, and Assignment 2, which involved the creation of list elements.

Once again, the constructs mentioned in the error messages give a sense of the context in which students encountered these errors. In Table IX, parent-child combinations that occurred more than once are shown.

Many of these errors involved description list (`dl`) elements and their required child elements `dt` and `dd`. The appearance of description lists is expected, since they were necessary for Assignment 2 and one of the first elements introduced in the course that required coordination with child elements. Ordered (`ol`) and unordered lists (`ul`) were similarly troublesome for students, particularly when nesting lists and sublists. In these cases, a common error was placing the opening sublist tag outside of its parent's list items.

Most of the remaining parent-child errors occurred when students nested block elements within inline elements. In HTML, there are two basic content models: block elements (e.g., `div`, `table`, `p`) expand to take up the available width, while inline elements (e.g., `span`, `strong`, `em`) contract around a text string. It is valid for block elements to have either block or inline elements as children, but with some exceptions, inline elements can only contain text or other inline elements. The distinction between block and inline elements (and their nesting rules) was carefully introduced in the course, but the extent to which students attended to these concerns while engaged in a given coding task is an open question.

*4.1.4. Other Errors.* Several of the remaining error types were organized around additional concepts. Parsing errors like "Saw < when expecting an attribute name" and "A slash was not immediately followed by >" indicated problems students had with the syntax within markup tags rather than between them. The syntax of void elements such as line breaks and horizontal rules, which are comprised of a single tag instead of a tag pair, also presented difficulties for the students, resulting in the "Self-closing syntax (/>) used on a nonvoid HTML element" error message. The occasional occurrence may be a typographical error; however, a persistent recurrence may be a warning sign, indicating deeper difficulties grasping the syntax of markup tags.

A second group of errors related to various representations of data, including HTML character references (e.g., `&copy;`), colors in hexadecimal notation (e.g., `#53A5C5`), and URLs (e.g., `http://openhtml.org`). Each of these introduces additional syntaxes that must be learned by students in order to properly format values.

### 4.2. Resolving Errors

The previous section reveals errors that remained in the final versions of students' assessments. In this section, we use validation as a lens for analyzing coding activity *during* the construction of web pages. We determine the proportion of syntax errors that are eventually resolved, how many validation attempts it takes students to correct them, and the role validation plays in helping to solve these errors. This is accomplished by analyzing the syntax errors present in the students' code during validation attempts and comparing them to the unresolved errors reported in the previous section. This analysis is based on a closer inspection of Assignments 1 and 2 for Spring 2013, enabled by the validation and logging features that were added to openHTML.

*4.2.1. Validator Usage.* On average, students made 12.6 validation attempts (SD = 15.0) across the two assignments. The extent to which the validators were used varied considerably, ranging from three students who did not use the validator at all to one student who used it 56 times. About half of validations (50.6%) resulted in one or more errors.

Based on observations of coding activity using the openHTML replayer, there seemed to be little correlation between validator usage and the ability to write syntactically correct code. Among students who showed the ability to write error-free code, some rarely validated their code until it was nearly completed while others used it methodically from early on. Similarly, among students who had more substantial difficulties, some relied on the validators heavily to debug their code while others used them rarely or not at all. This is characteristic of the behavioral differences in stoppers, movers, and tinkerers that have been observed among students struggling with Java syntax errors [Jadud 2006].

*4.2.2. Recurrence.* Validations generated 582 total errors comprising 23 error types. Identical errors, determined by error message and location in consecutive validations, were combined into a single episode, resulting in 268 episodes. These are summarized in Table X. Of these episodes, 27.6% related to nesting errors and 28.7% to parent-child rules, which is roughly in line with the unresolved errors that were observed in the previous section.

In addition to frequency and prevalence, recurrence was calculated as the number of consecutive validation attempts that a particular error lasted. Recurrence indicates how persistent an error is and suggests the degree with which students had trouble resolving it through repeated validation attempts. Errors requiring multiple validation attempts are suggestive of deeper conceptual difficulties as opposed to typographical mistakes and other slips.

40.7% of errors lasted more than one validation, up to a maximum of seven. The mean recurrence rate was 2.0 validation attempts (SD = 1.5) and the median was 1. Since the recurrence of errors was highly skewed, with most lasting only one validation attempt, the median and maximum are provided in the preceding table. One explanation for this skew is that fixing one validation error commonly resolved several additional errors. Given that most errors lasted only one validation attempt, comparing error types by

Table X. Types of Errors Made during Validation Attempts
Frequency is the number of episodes of an error, prevalence is the number and percentage of students that made an error at least once, recurrence includes the median and maximum number of validations that an episode lasted, and resolution is the number and percentage of episodes that were eventually resolved

| Error Type | Frequency | Prevalence | Recurrence Median | Recurrence Max | Resolution |
|---|---|---|---|---|---|
| Element Y is not allowed as child of element X in this context. | 73 | 10 (77%) | 1 | 7 | 65 (89%) |
| Consecutive hyphens did not terminate a comment. – is not permitted inside a comment, but e.g. - - is. | 39 | 3 (23%) | 4 | 6 | 35 (90%) |
| Unclosed element X | 23 | 7 (54%) | 1 | 3 | 22 (96%) |
| End tag X | 21 | 4 (31%) | 3 | 5 | 21 (100%) |
| Attribute Y not allowed on element X at this point | 20 | 1 (8%) | 1 | 1 | 20 (100%) |
| Named character reference was not terminated by a semicolon (or & should have been escaped as &.) | 17 | 4 (31%) | 1 | 3 | 17 (100%) |
| No X element in scope but an X end tag seen | 16 | 5 (38%) | 2 | 2 | 16 (100%) |
| End tag for X seen, but there were unclosed elements | 13 | 7 (54%) | 1 | 3 | 12 (92%) |
| End tag X seen, but there were open elements | 10 | 4 (31%) | 1 | 1 | 10 (100%) |
| Y is not an X value. | 4 | 2 (15%) | 1 | 2 | 4 (100%) |
| Stray end tag X | 4 | 2 (15%) | 1 | 1 | 4 (100%) |
| Property X does not exist. | 4 | 2 (15%) | 1 | 2 | 4 (100%) |
| Parse error | 4 | 1 (8%) | 1 | 2 | 4 (100%) |
| End tag X violates nesting rules. | 4 | 1 (8%) | 1 | 1 | 4 (100%) |
| Bad value Z for attribute Y on element X | 4 | 2 (15%) | 1 | 1 | 4 (100%) |
| Garbage after </ | 3 | 1 (8%) | 3 | 5 | 3 (100%) |
| Text not allowed in element X in this context | 2 | 1 (8%) | 2 | 2 | 2 (100%) |
| Element X is missing a required child element. | 2 | 2 (15%) | 1.5 | 2 | 2 (100%) |
| Saw = when expecting an attribute name. Probable cause: attribute name missing | 1 | 1 (8%) | 1 | 1 | 1 (100%) |
| No space between attributes | 1 | 1 (8%) | 1 | 1 | 1 (100%) |
| Character reference was not terminated by a semicolon. | 1 | 1 (8%) | 2 | 2 | 1 (100%) |
| A slash was not immediately followed by >. | 1 | 1 (8%) | 1 | 1 | 1 (100%) |
| < in attribute name. Probable cause: > missing immediately before | 1 | 1 (8%) | 1 | 1 | 1 (100%) |

their average recurrence rates is not highly instructive. Instead, recurrence seems most useful as a means for detecting cases when an individual is having acute difficulties, expressed as high maximum recurrence rates in the preceding table.

We note that recurrence rates alone do not fully capture the extent of difficulties students had resolving a particular problem. Inspecting their coding activity through the openHTML replayer revealed that students often engaged in tinkering, toggling the faulty code or eliminating it completely but getting no closer to a solution. At other times, one mistake led to a cascade of error messages that were all addressed by fixing the original error. In these situations, recurrence rates would be low and not accurately reflect the scope of the problem.

*4.2.3. Resolution.* Despite many HTML and CSS syntax errors lasting multiple validation attempts, students were eventually successful in resolving them in nearly all cases. Among the 268 errors detected in Assignments 1 and 2, only 14 were unresolved in the final submissions—a 94.8% rate. This was consistent from one error type to the next, most ranging between 90% and 100%.

Given these high resolution rates, we sought an explanation for the quantity of unresolved errors found in the first part of our analysis. We discovered that in addition to the 14 unresolved errors reported here, 67 were introduced after the final validation attempt or in code that was never validated at all. In other words, 82.7% of the unresolved errors were never brought to the attention of students through validation. It is likely that a similarly high proportion of unresolved errors in the other assessments were never fixed due to a failure to validate.

## 5. DISCUSSION

### 5.1. Mastering Syntax through Practice

In this study, we focused on syntax errors in HTML and CSS, two languages that are fundamental to web development but often overlooked in computing education research. Our results highlight that despite the seeming simplicity of these languages, their syntaxes can present many challenges for beginners. On average, students had 16.6 unresolved errors across the six assessments included in our analysis; only four students submitted error-free code. These errors not only present obstacles to authoring syntactically valid code, but also compound difficulties with semantics and design.

Accounting for nearly a quarter of the unresolved errors were issues related to parent-child rules. Parent-child relationships follow an extensive system of rules that govern when it is valid for certain types of HTML elements to be nested in one another. With the introduction of new elements, parent-child rules and the interactions between them continue to grow. In the study, this was reflected in the types of errors students made as they progressed, which ballooned with the introduction of lists and sublists. We expect that as students learn other compound elements such as tables and forms, and work with larger, more complex web pages, they will continue to make errors that violate the myriad parent-child rules.

On the other hand, the syntax for nesting tags is relatively simple and consistent from one element to the next, yet it accounted for over one-third of unresolved errors. Although students appeared to grasp the syntax of nesting tags quickly (all of the students demonstrated proper nesting from the first assessment), errors related to nested tags occurred with regularity during all five weeks of the study. These errors often manifested when students were confronted with compound elements like lists, and deeper levels of nested code. This suggests that these nesting errors were attentional in nature—as they grappled with unfamiliar or complex code and their cognitive load was taxed [Chandler and Sweller 1996], an end tag was forgotten or misplaced.

What this underscores is that beyond declarative knowledge, practice plays an important role when learning HTML and CSS. The syntax of nesting markup tags may be learned on day one, but the development of skills related to reading and writing nested code—which includes visually parsing delimiters (start and end tags in the case of HTML), and mentally translating them into a hierarchical structure—is an ongoing process. Through practice, the deliberate activity of reading and writing nested code can eventually become highly routinized skills [Rasmussen 1983], helping minimize these errors while freeing cognitive load for higher-level concerns. This parallels research on reading text, where reading skills at the letter and word levels have been found to influence higher-level reading comprehension and achievement [Biemiller 1977].

### 5.2. Learning through Validation

Our findings show that validation is important, surfacing syntax errors in half of the students' validation attempts. Furthermore, validation is effective, evidenced by the eventual resolution of nearly 95% of the errors detected with validation. In contrast, most of the unresolved errors (83%) were present in code that was never tested. Despite its effectiveness, many students did not validate their code. This is all the more surprising given that validation was an explicit requirement of the homework assignments, and that in later weeks, openHTML's integrated validator provided added convenience.

One of the challenges of web development is that validation is optional, unlike programming languages like Java requiring a compilation step. Complicating matters further, HTML is a highly forgiving language by design, and browser engines attempt to render a web page even in the presence of syntax errors, implicitly modifying the source code if necessary to do so. This results in cases where a web page displays exactly as intended while numerous syntax errors remain latent in the source. Beyond the quality of the code itself, this lack of feedback can lead to the development of poor habits and faulty mental models that do not equip students with the ability to predict the relationship between input and output in new contexts [Du Boulay 1986].

Beyond teaching validation practices specifically and testing more generally, there is an opportunity to encourage validation through the design of web editors. For instance, displaying the validation status of a web page upon saving it may help users to maintain awareness of latent errors in code and motivate users to correct them. Many existing web editors go even further, providing instant feedback of errors detected in the code.

Finally, there are opportunities to improve validator feedback by addressing understanding and suggesting solutions. Although students successfully corrected most of the errors detected during validation, there were cases that required numerous validation attempts. The feedback provided by the validators likely contributed to the difficulties students had in resolving these errors.

Many errors generated cryptic feedback; programming language compiler feedback is likewise known to cause novices trouble [Nienaltowski et al. 2007]. This was especially the case for the CSS validator, which returned terse "parse error" or "value error" messages. Furthermore, one message could be associated with multiple, disparate errors. A common error, "element Y not allowed as child of parent element X in this context," was made at least once by 77% of students, with 65% leaving one or more of them unresolved. This error occurred in two distinct circumstances: when parent-child rules were violated by valid elements and when invalid elements (e.g., `<text>` and `<stong>`) were used at all. Conversely, one error could be associated with multiple error messages. An extraneous end tag could alternately trigger a "stray end tag" or "no element in scope but end tag seen" message depending on the context in which it occurred. Students had little help in understanding the reasons for these distinctions.

When a student is validating their code and reviewing their errors, this presents a critical learning opportunity. Rather than mere technical correctness, we see these events as a chance to provide actionable feedback that helps students learn to author correct code and improve their understanding [Hartmann et al. 2010].

### 5.3. Limitations

Several limitations temper our findings. First, this study focuses on a relatively small sample of students in a single course. Therefore, while our findings shed light on the types of syntax errors novices make, they are not representative of web development students in all contexts. For example, participants in our study were mostly non-CS majors with minimal prior programming experience. It is possible that, for instance, a class comprised of CS majors with significant programming experience would commit fewer syntax errors related to nesting tags, which draws on general skills associated with program composition and comprehension [Corritore and Wiedenbeck 1991], compared to parent-child rules that are more specific to the domain of web development.

Moreover, this study focused on early assignments that introduced features of HTML and a small amount of CSS. We did not track student activity involving JavaScript and more complex HTML and CSS, which are likely to introduce different kinds of errors and resolution strategies. This was due to the limited viability of openHTML in later weeks of the course. openHTML achieves much of its simplicity by supporting only single web pages. The drawback of this approach is that reusing a CSS style sheet or other resources between multiple HTML documents becomes cumbersome, requiring users to duplicate their efforts for each webpage. The instructor of the course was able to use openHTML for the two-page site in Assignment 2 through the careful design of the assignment and guidance for the students, but using openHTML beyond this point was not feasible.

Our log analysis provided us with a fine-grained view of coding behavior, but we were limited by a lack of contextual clues compared to our earlier lab study [Park et al. 2013b]. Although the assignments provide some guidance on the students' overall objectives, this lack of context limited our ability to infer their intent with respect to more granular actions. Because of these limitations, we focused our analysis on syntax errors, setting aside difficulties they might have planning the design of a web page or semantic errors they may have made while creating it. These also comprise a significant portion of their learning experience and have the potential to impact their attitudes and progress in learning web development as much as syntax errors.

Finally, this study made error messages the unit of analysis. As we noted in our results, multiple errors often tied together as extended episodes of debugging or as symptoms of a deeper conceptual problem. How this data can be effectively interpreted to interpret higher-order understanding is an open question for future research. One potential approach is to couple remote log analysis with more direct methods of inquiry. For instance, in a follow-up study, students might be presented with errors they have made in the course and asked to interpret and correct them.

## 6. CONCLUSIONS AND FUTURE WORK

This study presents our analysis of the coding activity of students in the initial weeks of a web development course. It extends work on novice programming errors and compilation behavior to the domain of web development, offering new insights into how beginners learn HTML and CSS. The contribution of this work is a detailed characterization of the errors students made with the syntax of HTML and CSS as they completed in-class and homework assessments. We found that nested tag errors made up a significant proportion of their troubles, and that they remained consistent weeks into the course. Parent-child rules governing when elements can be nested were also problematic for students. Our analysis also highlights the critical role validation plays in how beginners resolve these errors. Most unresolved errors occurred in untested code, while the vast majority of errors that were detected through validation were eventually corrected.

In future work, we hope to extend our study for the duration of an introductory web development course in order to investigate how students learn CSS and more complex HTML. We also plan to explore how analytical methods like the ones presented in this article can be used to detect cases of severe difficulties and predict performance in a course. Finally, we will continue work on designing coding environments to provide better support for beginners in avoiding, correcting, and understanding errors as they learn web development.

## ACKNOWLEDGMENTS

Our heartfelt thanks to the students who participated in this study, and to the National Science Foundation for their support.

## REFERENCES

A. Biemiller. 1977. Relationships between oral reading rates for letters, words, and simple text in the development of reading achievement. *Read. Res. Q.* 13, 2 (1977), 223–253.

A. 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*. 2–10.

J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer. 2009. Two studies of opportunistic programming: Interleaving web foraging, learning, and writing code. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1589–1598.

K. Brennan and M. Resnick. 2012. New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the Annual Meeting of the American Educational Research Association*. 1–25.

P. Chandler and J. Sweller. 1996. Cognitive load while learning to use a computer program. *Appl. Cognitive Psych.* 10, 151–170.

R. Connolly. 2011. Awakening Rip Van Winkle: Modernizing the computer science web curriculum. In *Proceedings of the Annual Conference on Innovation and Technology in Computer Science Education*. 18–22.

C. L. Corritore and S. Wiedenbeck. 1991. What do novices learn during program comprehension? *Int. J. Hum.-Comput. Int.* 3, 2 (1991), 199–222.

A. Désilets, S. Paquet, and N. G. Vinson. 2005. Are wikis usable? In *WikiSym*. 3–15.

B. Dorn and M. Guzdial. 2010a. 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*. 703–712.

B. Dorn and M. Guzdial. 2010b. Discovering computing: Perspectives of web designers. In *Proceedings of the International Computing Education Research Workshop*. 23–29.

B. Du Boulay. 1986. Some difficulties of learning to program. *J Educ. Comput. Res.* 2, 1 (1986), 57–73.

T. Felke-Morris. 2011. *Basics of Web Design: HTML5 and CSS3*. Addison-Wesley, Boston, MA.

S. Fincher, J. Tenenberg, and A. Robins. 2011. Research design: Necessary bricolage. In *Proceedings of the International Computing Education Research Workshop*. 1–6.

B. Hartmann, D. MacDougall, J. Brandt, and S. R. Klemmer. 2010. What would other programmers do? Suggesting solutions to error messages. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1019–1028.

M. C. Jadud. 2006. Methods and tools for exploring novice compilation behaviour. In *Proceedings of the International Computing Education Research Workshop*. 73–84.

F. Klassner. 2000. Can web development courses avoid obsolescence? In *Proceedings of the Annual Conference on Innovation and Technology in Computer Science Education*. 77–80.

B. B. L. Lim. 1998. Teaching web development technologies in CI/IS curricula. In *Proceedings of the ACM Technical Symposium on Computer Science Education*. 107–111.

R. Mercuri, N. Herrmann, and J. Popyack. 1998. Using HTML and JavaScript in introductory programming courses. In *Proceedings of the SIGCSE Technical Symposium on Computer Science Education*. 176–180.

C. S. Miller, L. Perkovic, and A. Settle. 2010. File references, trees, and computational thinking. In *Proceedings of the Annual Conference on Innovation and Technology in Computer Science Education*. 132–136.

M. Nienaltowski, M. Pedroni, and B. Meyer. 2007. Compiler error messages: What can help novices? In *Proceedings of the SIGCSE Technical Symposium on Computer Science Education*. 168–172.

S. Papert and I. Harel. 1991. Situating Constructionism. In *Constructionism.* S. Papert and I. Harel (Eds.). Ablex Publishing Corporation, 1–13.

T. H. Park and S. Wiedenbeck. 2011. Learning web development: Challenges at an earlier stage of computing education. In *Proceedings of the International Computing Education Research Workshop*. 125–132.

T. H. Park, R. M. Magee, S. Wiedenbeck, and A. Forte. 2013a. Children as webmakers: Designing a web editor for beginners. In *Proceedings of the Conference on Interaction Design and Children*. 419–422.

T. H. Park, A. Saxena, S. Jagannath, S. Wiedenbeck, and A. Forte. 2013b. Towards a taxonomy of errors in HTML and CSS. In *Proceedings of the International Computing Education Research Workshop*. 75–82.

R. D. Pea, E. Soloway, and J. C. Spohrer. 1987. The buggy path to the development of programming expertise. *Focus on Learning Problems in Mathematics* 9, 1 (1987), 5–30.

J. Rasmussen. 1983. Skills, rules, and knowledge; Signals, signs, and symbols, and other distinctions in human performance models. *IEEE Trans. Syst., Man, Cybern.* 13, 3 (1983), 257–266.

D. Reed. 2001. Rethinking CS0 with JavaScript. In *Proceedings of the SIGCSE Technical Symposium on Computer Science Education*. 100–104.

M. B. Rosson, J. F. Ballin, and H. Nash. 2004. Everyday programming: Challenges and opportunities for informal web development. In *Proceedings of the Symposium on Visual Languages and Human-Centric Computing*. 123–130.

M. B. Rosson, J. F. Ballin, and J. Rode. 2005. Who, what, and how: A survey of informal and professional web developers. In *Proceedings of the Symposium on Visual Languages and Human-Centric Computing*. 199–206.

J. C. Spohrer and E. Soloway. 1986. Novice mistakes: Are the folk wisdoms correct? *Commun. ACM* 29, 7 (1986), 624–632.

K. Sridharan. 2004. A course on web languages and web-based applications. *IEEE Trans. Educ.* 47, 2 (2004), 254–260.

A. Stefik and S. Siebert. 2013. An empirical investigation into programming language syntax. *ACM Trans. Comput. Educ.* 13, 4 (2013), 1–40.

K. Treu. 2002. To teach the unteachable class: An experimental course in web-based application design. In *Proceedings of the SIGCSE Technical Symposium on Computer Science Education*. 201–205.

E. L. Walker and L. Browne. 1999. Teaching web development with limited resources. In *Proceedings of the SIGCSE Technical Symposium on Computer Science Education*. 12–16.

Y. D. Wang and N. Zahadat. 2009. Teaching web development in the Web 2.0 era. In *Proceedings of the ACM Special Interest Group for Information Technology Education*. 80–86.

C. Watson, F. W. B. Li, and J. L. Godwin. 2014. No tests required: Comparing traditional and dynamic predictors of programming success. In *Proceedings of the ACM Technical Symposium on Computer Science Education*. ACM, New York, New York, 469–474.

S. Wiedenbeck. 2005. Factors affecting the success of non-majors in learning to program. In *Proceedings of the International Computing Education Research Workshop*. 13–24.

World Wide Web Consortium. 2014. Why Validate? Retrieved from http://validator.w3.org/docs/why.html.

Received September 2013; revised November 2014; accepted November 2014