Many articles, like Nate Zelnick’s Standards column (“Nifty Technology and Nonconformance: The Web in Crisis,” Computer, Oct. 1998, pp. 115-116, 119), attribute the Web’s continuing development crisis to the failure of commercial browsers to fully implement agreed-upon standards. This is an important issue: Nonconformant and incomplete implementations have been a nightmare for Web developers. However, there is a deeper issue—one that has had little public discussion: Will these standards actually provide the envisioned benefits for the Web?

Style sheets, which specify how documents are presented to users, are expected to play a critical role in the Web’s architecture. Through the use of style sheets, future Web documents will be easier to author and will be accessible everywhere, from PCs to TVs to palm devices to cellular phones. Unfortunately, the current standards appear to impede this grand vision of the Web.

The Vision

Visionaries foresee a Web in which pages have rich semantics that enhance users’ understanding and also facilitate searching with automated tools. In this grand vision, a huge variety of devices can view these semantically dense pages in innumerable styles tuned to users’ diverse tastes, needs, and interests. This improved Web will offer total accessibility, with users listening to Web pages through aural browsers that speak or feeling them through Braille devices.

Style sheets are a critical part of this vision. The Extensible Markup Language (XML), which is expected to replace HTML, allows Web documents to contain much richer semantic information than is possible with HTML. However, unlike HTML, XML doesn’t specify anything about a document’s appearance. Browsers get this missing information from style sheets, which are separate appearance specifications. Because the style sheets are separate, it should be possible to mix and match them with documents to provide a variety of display and playback styles. An author only has to create a single version of a page, and users can apply appropriate style sheets to adapt the page to their devices and needs.

This vision of Web accessibility imposes a number of requirements. First, style sheet languages and systems must support the complete separation of content (XML) and presentation (style sheets) so that it isn’t necessary to change content to suit the presentation’s needs. Second, users must have final say over how style sheets are applied because they know their own needs best. This implies that final control over the style sheets will reside on the client side. Third, to have control over presentation, end users must be able to write at least the more simple style sheets themselves. This means that the languages must be relatively easy to use, but they must also have sufficient expressive power to support a wide range of complex presentations.

Style Sheet Standards

W3C has two style sheet standards: Cascading Style Sheets (CSS) for HTML and the Extensible Style Language (XSL) for XML. Although CSS’s architecture appears to be well suited to the Web, its language contains flaws that would hinder good engineering practices even if it were fully supported by browsers. However, XSL, the Web’s next-generation style sheet standard, appears to contain even more significant flaws.

Cascading Style Sheets

On the positive side, CSS provides moderately good support for authors and end users to specify how Web documents are presented on a variety of devices, including aural browsers. CSS’s overall architecture satisfies the requirement of giving users final say over how style sheets are applied: CSS defines how to cascade or combine style sheets supplied by users and clients with style sheets from authors. In addition, CSS provides easy access to common style solutions for HTML.
Simple style sheets are almost trivially easy to write. For example, \( \texttt{P (font-size:12pt;)} \) is a complete style sheet specifying that paragraphs (P elements) should use a 12-point font. So, like HTML, CSS has a gradual learning curve.

On the negative side, CSS's language has some flaws: Its semantics are inconsistent, and it has limited expressive power. In CSS, the property rules have the form \(<\text{property}> : <\text{value}>\). The set of acceptable values varies substantially from property to property. Furthermore, a value's interpretation can also vary from property to property. For example, \( \text{font-size:80\%} \) means 80 percent of the parent element's font size, while \( \text{line-height:80\%} \) means 80 percent of the same element's font size. Even though the semantics for each property are generally quite intuitive, as properties are added to CSS, we believe its users will be overwhelmed by special cases, and it will be harder to understand the language.

Unfortunately, CSS doesn't allow its users to specify values via arbitrary mathematical expressions. As a result, style sheet authors can express only those cases that CSS's designers have recognized as valuable. For example, there is no way to set an element's font size to 80 percent of the root element's font size, which might be expressed as \( \text{font-size:root*80\%} \). CSS's lack of expression support is particularly crippling for layout, where mathematical expressions are a powerful tool for positioning elements relative to one another. Other limitations on CSS's expressive power include restrictions on contextual selectors and the fact that generated material does not support the full range of style properties.

These limits on expressive power restrict the styles authors can create, and they also hinder the separation of presentation and content. When a style effect can't be described in a general way, Web developers inevitably embed style information in the source document that contains the content, using ID tags or presentation tags to get the desired effects. These tricks still require using a style sheet, but the developers use style sheets specialized for a single document, rather than sharing a style among many documents. Thus, they lose one of the most important engineering benefits of style sheets.

**Extensible Style Language**

We are skeptical that XSL can fulfill its envisioned role as XML's style sheet companion. XSL is a much larger language than CSS, and it defines a formatting vocabulary within the larger framework of a transformation language. Most XSL standardization efforts have focused on its transformation language. On the positive side, XSL supports separation of presentation and content, and it has considerable expressive power, including the ability to handle arbitrary mathematical expressions.

Unlike CSS's architecture, XSL's architecture doesn't allow end users or their clients to supply style sheets to control document presentation. While the specification doesn't explicitly preclude user-provided style sheets, it also doesn't directly address the issue. This issue's importance can't be overemphasized: Client-supplied style sheets are critical to the goal of making the Web accessible from any device by any user. Without client-supplied style sheets, users have to depend on Web sites to support their devices and special needs. We doubt that large numbers of Webmasters would choose to publish versions for low-usage clients like aural and Braille devices, and we are certain that they can't anticipate every new browsing device's characteristics.

XSL is directly derived from the strongly page-based Document Style Semantics and Specification Language (DSSSL) for SGML documents, and XSL has inherited DSSSL's page-oriented formatting model. In fact, in XSL's current draft, even aural presentations are required to contain a sequence of pages. But why would an aural presentation have pages? Aural presentation has different needs than print presentation, and it demands a more flexible formatting approach. Thus, XSL's page-oriented approach limits its suitability for the full range of anticipated browsing devices.

While there is considerable debate about what makes a language “easy to use,” we find using XSL difficult. First, because it uses XML syntax, XSL is expressed through markup tags, such as \(<\text{xsl:apply-templates/>}\) and \(<\text{fo:simple-page-master}>\). This is an unusual, and possibly inappropriate, use of the markup paradigm because markup is typically used to annotate data with metadata, but XSL style sheets are almost entirely metadata (style and transformation instructions). Furthermore, this tag-based syntax means that XSL is verbose.

A second difficulty with XSL derives from its declarative transformation language. Document transformations arrange the material in a document in the order that it will be laid out on the page or screen. XSL advocates believe that its declarative approach is fundamentally easier to use than the imperative approach of scripting or programming languages. We disagree. Declarative languages are only easy to use when the user doesn't need to understand the language's underlying processing model. But, to write an XSL style sheet effectively, you need to understand processing models for both tree transformation and formatting.

Finally, XSL has a steep learning curve. Unlike CSS, simple XSL style sheets require substantial amounts of code. CSS avoids this problem through a combination of browser-defined defaults and its cascading mechanism.

**UNEXPLORED TERRAIN**

Style sheet languages are terribly under-researched. Only a few style sheet languages have been designed and implemented, and there is little accepted wisdom about the requirements for style sheet systems. The lack of solutions makes it difficult to make informed choices about the most appropriate style sheet language for the Web's grand vision.

In CSS's case, this lack of research has been less damaging because the CSS com-
committee had a well-understood goal of improved style control for HTML, and they started with the style services used in everyday word-processing applications. The lack of prior research has been a much greater problem in the search for XML style solutions. XSL's development has been heavily based on DSSSL. At first glance, this seems logical since DSSSL is the ISO style sheet companion for SGML, and XML is derived from SGML. But DSSSL and XSL have a far narrower vision. They primarily address the needs of high-volume producers of print documents, such as government agencies. The Web's end users have very different needs.

Our research into style sheet systems and languages (Proteus and PSL) shows that there are alternatives to CSS and XSL that more closely meet the above criteria. The PSL style language has a syntax especially designed for the style sheet task and has traditional computational features including mathematical expressions and conditionals. In contrast to XSL, PSL does not emphasize transformations, and it uses constraints, rather than flow, to specify layout. Furthermore, there are other style languages, such as the Thot structured document toolkit's P language and the more recent Constraint CSS.

To while we realize that substantial resources have been invested in CSS and XSL, we nevertheless urge the Web community to more fully explore alternate approaches to style sheets.

Philip M. Marden, Jr., is a master's student in computer science at the University of Wisconsin-Milwaukee. Contact him at pmarden@acm.org.

Ethan V. Munson is an assistant professor in the Department of Electrical Engineering and Computer Science at the University of Wisconsin-Milwaukee. Contact him at munson@cs.uwm.edu.