The One Hundred Year Web

Steven Pemberton
Centrum Wiskunde & Informatica Amsterdam, The Netherlands
steven.pemberton@cwi.nl

ABSTRACT

The year 2023 marks the thirty-second anniversary of the World Wide Web being announced.

In the intervening years, the web has become an essential part of the fabric of society. Part of that is that huge amounts of information that used to be available (only) on paper is now available (only) electronically. One of the dangers of this is that owners of information often treat the data as ephemeral, and delete old information once it becomes out of date. As a result society is at risk of losing large parts of its history.

So it is time to assess how we use the web, how it has been designed, and what we should do to ensure that in one hundred years time (and beyond) we will still be able to access, and read, what we are now producing. We can still read 100 year-old books; that should not be any different for the web.

This paper takes a historical view of the web, and discusses the web from its early days: why it was successful compared with other similar systems emerging at the time, the things it did right, the mistakes that were made, and how it has developed to the web we know today, to what extent it meets the requirements needed for such an essential part of society's infrastructure, and what still needs to be done.

CCS CONCEPTS

• **Social and professional topics** → Professional topics; History of computing; History of software; • **Information systems** → World Wide Web; Web data description languages; Markup languages; • **General and reference** → Cross-computing tools and techniques; Design.

KEYWORDS

World Wide Web, History, Design, Declarative principles, Markup, HTML, XML, XHTML, HTML5, Ephemera, Longevity, Data conservancy

ACM Reference Format:
Steven Pemberton. 2023. The One Hundred Year Web. In *Companion Proceedings of the ACM Web Conference 2023 (WWW '23 Companion), April 30–May 04, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3543873.3585578

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs International 4.0 License.

WWW '23 Companion, April 30–May 04, 2023, Austin, TX, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9419-2/23/04.
https://doi.org/10.1145/3543873.3585578

1 THE WEB OF THE LONG NOW

New College, Oxford, built in 1379, has a dining hall with huge oak beams in the roof. At a given point, they discovered the beams needed replacing. But where do you find oak beams? So they approached the University forester, and asked him.

> *"Which college are you from?"* he asked, *"New College"* they replied. *"Well, I've got your trees"*.

It turns out that around the time that New College was built, they planted new trees to be ready for when they would need them [3].

We don't see that sort of attitude much these days.

This paper addresses the history and development of the web, how it has progressed, and what we need still to do, amongst other things to ensure longevity of its content.

2 THE ORIGINAL WEB

The year 2023 marks the thirty-second anniversary of the World Wide Web being announced: on 6 August 1991, Tim Berners-Lee posted a short summary of the World Wide Web project to an internet newsgroup inviting collaborators; the first web servers had been made publicly available a few months earlier [9] .

The web had been made possible by the internet becoming open and international, after the first open internet node outside of North America was installed at the CWI in Amsterdam, the Netherlands in November 1988 [6]. Two spin-offs were created to extend the internet into the rest of Europe. On that day in 1988, public computing itself was barely thirty years old: in 1957 a computer had been installed for the first time in a municipality, as it happens in Norwich, UK [16].

In many ways, the original web was not revolutionary: hypertext programs that could do similar things already existed. What the original web did was create the right combination of existing elements:

- a hypertext foundation
- connected to the internet
- using a simple and easy-to-implement protocol very similar to the existing FTP protocol
- separating document formats from the delivery methods
- using an existing markup methodology (SGML) that was easy to understand and use, even for non-technical people
- allowing the combination of existing internet delivery mechanisms and formats so that people could leverage their existing online content for their first web site,

and possibly the most important one:

- giving it all away for free.

The major innovation was the URL [17], which allowed you to combine documents from a plethora of sources into a single web page.

Another major property was that it was based on *declarative* principles.

3 THE DECLARATIVE PRINCIPLE

We learn in school what numbers are, and how to add, subtract, multiply and divide them. These are all procedural methods. However, when we get to square roots, we are only told:

> *The square root of a number is another number that multiplied by itself gives the original number.*

This is a *declarative* definition. It tells you what something is, it tells you how to recognise it, but it doesn't tell you how to calculate it. Most people know what a square root is, but very few people leave school knowing how to calculate one.

Now consider a procedural definition of square root:

```
function f a:
{
   x ← a
   x' ← (a + 1) ÷ 2
   eps ← 1.19209290e-07
   while abs(x − x') > eps × x:
   {
      x ← x'
      x' ← ((a ÷ x') + x') ÷ 2
   }
return x'
}
```

This definition raises many questions, not least of which is *What does it even do?* But other questions include: *Under what conditions does it work? How does it do it? What is the theory behind it? Is it correct? Can I prove it? Under what conditions may I replace it, or parts of it with something else?*

In fact, even if you know the theory, it is hard to determine how it is used in this code, because the code has been optimised by unrolling the loop once, and pre-evaluating some constant expressions. The issue is that the solution is very far from the problem statement; this is one of the reasons that documentation is so important in programming. In a nutshell, the advantages of the declarative approach are that it is:

- (Much) shorter
- Easier to understand
- Independent of implementation
- Less likely to contain errors
- Easier to see it is correct
- Tractable.

4 DECLARATIVE MARKUP

One of the strengths of the original web was its declarative markup: although there were some mistakes, the markup largely specified the role of the elements, rather than how they should appear. For instance, an h1 was a top-level heading with no *a priori* requirement that it be displayed in any particular way, larger or in bold. It just stated its purpose.

Mistakes included hr (horizontal rule), and elements like b and i for bold and italic, which specify a visual property rather than a purpose, but most of the structure was purely declarative.

This has a number of advantages, including machine and modality independence: you can just as easily 'display' such a document with a voice-reader as on a screen, without having to use heuristics to guess what is intended.

The poster-child of HTML declarative markup is the <a> element:

```
<a href="talk.html" title="..." target="..." class="...">
My Talk</a>
```

This single line compactly encapsulates a lot of behaviour including

- what the link looks like
- what to do when you hover over the link
- activating the link in several ways
- what to do with the result
- hooks for presentation changes.

Doing this procedurally in program code would be a *lot* of work.

5 STYLE SHEETS

Another advantage of declarative markup is that since display properties are not baked in to the language you can use style sheets to control the display properties of a document, without altering the document itself.

In fact one of the first activities of the newly-created W3C was to add style-sheets as quickly as possible to undo the damage being done by the browser manufacturers, who were unilaterally adding visually-oriented elements to HTML, such as font, and blink.

The result, CSS, is another example of a successful declarative approach [4].

When W3C started the CSS activity, Netscape, at the time the leading browser, declined to join, saying that they had a better solution, JSSS, based on Javascript – in other words a procedural rather than declarative approach. Instead of the declarative CSS

```
h1 { font-size: 20pt }
```

you would use script to say

```
document.tags.H1.fontSize = "20pt";
```

The entry on Wikipedia remarks:

> *"JSSS lacked the various CSS selector features, supporting only simple tag name, class and id selectors. On the other hand, since it is written using a complete programming language, stylesheets can include highly complex dynamic calculations and conditional processing."* [12]

6 IMPLEMENTERS AS DESIGNERS

Implementers tend in general not to be great designers, because of their tendency to focus on the implementation needs rather than the user needs. For example, the original HTML surprisingly did not have facilities for embedding images into documents, so they were added by the implementers of the first really successful browser, Mosaic.

Unfortunately, they didn't do a great job. They added a single element <img src="..."> to embed an image at that location in the code. This has two regrettable, related, disadvantages: firstly, there is no failure fallback, and secondly there is no alternative for non-visual environments.

A better design would have allowed the element to have content to be used in fallback cases. For instance

```
<img src="cat.png">
   <img src="cat.jpg">
      A <em>cat</em>, sitting on a mat.
  </img>
</img>
```

If the outer img should fail for whatever reason (the resource unavailable, the browser not supporting png images, or it being a non-visual browser), the nested img would be tried, and if that failed, the text would be used. The advantage of such a design to visually impaired users of the web should be obvious. When png images were introduced on the web, their usage was held back for a long time because of the lack of such a mechanism: authors had to wait until a critical mass of browsers were available that could display the new image type before they could start using them, creating the conditions for a potential vicious circle of them not being used because there were no implementations, and not being implemented because there were no users.

We have already mentioned the unfortunate blink and font elements that were introduced by the implementers, and we should not let that excrescence the frameset, with its security and usability problems, go unmentioned either.

7 HTML 4

One of the early tasks of the nascent W3C was to try and undo the damage being inflicted on the web by the implementors. By then there were two warring browsers both adding new things, often incompatible, and without consulting the community. The W3C result was HTML4, a compromise between the different browsers, but with a clear development path [10].

Examples of compromises that had to be made are HTML events which have have both a capture and bubble phase, since the two main browsers did it differently, and that the meta element not having content, but a content attribute instead, because one of the browsers incorrectly displayed content in the head.

HTML4 came in three versions:

- Strict, indicating the future direction, and disallowing many of the inappropriate elements;
- Transitional, in which the deprecated elements were allowed;
- Frameset, where frame elements were allowed.

HTML4 also properly used SGML, so that HTML documents could be read and produced by existing SGML processors. However, it was observed that SGML was overly complex for the task, and an activity was started to define a simpler version of SGML, a subset, which became XML [23].

8 XML

XML had a big advantage as markup language, namely that it would be possible to create documents that combined markup languages from different domains. This meant that domain experts could design (sub-)languages for their domain, that with proper design would be combinable with other markup languages. Examples of these domains included graphics (SVG), Mathematics (MathML), Multimedia (SMIL), and Forms (XForms), although there were other domains including semantics, and interaction events.

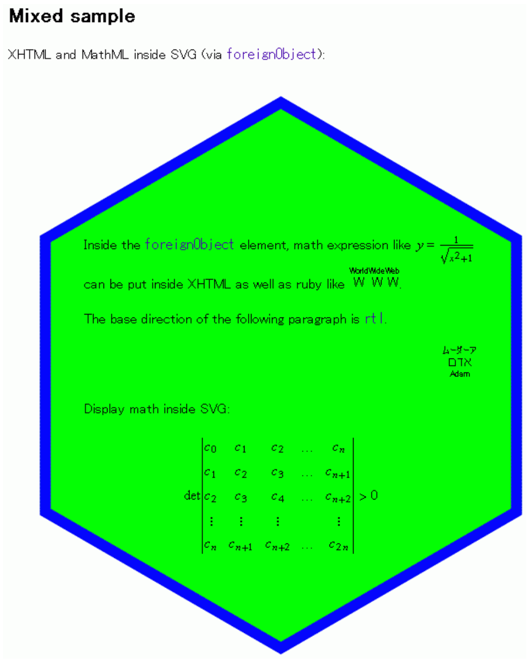
The advantages of such modularity should be obvious to anyone who has programmed: they allow you to specify a thin interface between the modules, and then design the modules independently. It also implied the need for an XML version of HTML, so that it could be part of this combinatorial activity, which became XHTML [24].

9 XHTML

The first version of XHTML was produced surprisingly quickly. There was wide-scale agreement on a need, and there were few decisions to be made, given that it was just to be a different serialization of the same structures from HTML4. There were similarly three versions as with HTML4, but with a clear indication that only *strict* would be further developed.

The fact that using XML allowed the mixing of namespaces in a single document was widely misunderstood, and there were complaints that XHTML added no elements, but it in fact added enormous amounts of functionality.

For instance, the image below is an example from 2002 of a single document (that ran in browsers already) combining XHTML, SVG and MathML [27] .



Mixed sample

XHTML and MathML inside SVG (via foreignObject):

Inside the foreignObject element, math expression like $y = \frac{1}{\sqrt{x^2+1}}$

can be put inside XHTML as well as ruby like $\overset{\text{WorldWideWeb}}{\text{W W W}}$.

The base direction of the following paragraph is rtl.

ムーダーア
אדם
Adam

Display math inside SVG:

$$\det \begin{vmatrix} c_0 & c_1 & c_2 & \dots & c_n \\ c_1 & c_2 & c_3 & \dots & c_{n+1} \\ c_2 & c_3 & c_4 & \dots & c_{n+2} \\ \vdots & \vdots & \vdots & & \vdots \\ c_n & c_{n+1} & c_{n+2} & \dots & c_{2n} \end{vmatrix} > 0$$

Also at this time there came a demand for variants of XHTML to serve particular needs. Notable examples are XHTML Basic [2], a smaller version for small devices such as mobile phones, and XHTML Print [19] for use with printers specifically for devices that were unable to load device drivers for printers.

To avoid problems of divergence, a modularisation mechanism for XHTML was devised [14], with corresponding modules, so that to define a new variant of XHTML, you only needed to specify which modules you needed, and you had your language, with guaranteed consistency across the variants, and if an error was later corrected in a module, all the variants that used that module would automatically be updated.

Consequently, a modularised version of XHTML was created, XHTML 1.1 [25], but only in a strict version, with only slight differences with XHTML 1.0 strict. This approach meant that when a version of XHTML was required with RDFa added, it was a simple matter of creating a module for RDFa, and publishing.

10 XHTML2

After XHTML 1.1, a new effort was started to make XHTML more consistent, clear up some historical glitches (such as the empty meta element, and no fall-back for images), and address new required functionality, such as better metadata and forms. This was to be XHTML2 [26].

Unfortunately, before it was ready, the group was closed by W3C management, despite the membership having voted for it to be retained.

The modules that the group were working on were moved to other groups to continue development, such as ARIA [1], RDFa [20], and XForms [22].

11 HTML5: A NEW WEB, BY PROGRAMMERS, FOR PROGRAMMERS

At that point HTML was taken on a completely different path, driven entirely by implementers, with little reference to users, predicated on procedural methods, disregarding the fundamental design principles of the web, and eschewing modularity, essentially turning HTML into a monolithic programming environment, namely HTML5 [11].

11.1 Design

Much of HTML5 is not designed in the normal sense of the word, although a design principles document was published [7].

One of the design principles quoted was "Pave the Cowpaths", "Cowpaths" being a rather derogatory term for what in design circles is normally referred to as "Desire Paths". This is a design-principle used in architecture: when you build a campus or estate, don't pave the paths, but wait and see where people actually walk, so you can see where they *need* paths.

But the design document got it wrong:

> *"When a practice is already widespread among authors, consider adopting it rather than forbidding it or inventing something new. Authors already use the <br/> syntax as opposed to <br> in HTML and there is no harm done by allowing that to be used."*

This however is not "Paving the cowpaths", which would be more like noticing that huge numbers of sites have a navigation drop-down, and supporting that natively.

But even "Paving the cowpaths" is not necessarily a good design practice in itself. Cows are not designers. Cowpaths are data. If you pave cowpaths, you are setting in stone the behaviours caused by the design decisions of the past. Cowpaths tell you *where* the cows want to go, not *how* they want to get there. If they have to take a path round a swamp to get to the meadow, then maybe it would be a better idea to drain the swamp, or build a bridge over it, rather than paving the path they take round it.

Paving cowpaths is a bad design principle in the way that it was applied. It can be a good design principle, but they apparently misunderstood it.

11.2 Faulty Cowpath-based Design

As an example, the HTML5 group spidered millions of pages, and then on the basis of that data decided what should be excluded from HTML5. This is exactly the opposite to "paving the cowpaths": it is putting fences across cowpaths that are used by fewer cows than some other paths, and even goes against their own proclaimed design principles.

As an example, take the @rev attribute.

```
<link rel="next" href="chap2.html"/>
<link rev="prev" href="chap2.html"/>
```

@rel and @rev are complementary attributes, they are a pair, like +/-, up/down, left/right.

The HTML5 group decided that not enough people were using @rev, and so removed it. This breaks backwards compatibility, and puts a fence before those who do need to use it. This is doubly bad in the light of another of their design principles: "Support Existing Content".

11.3 Irritated by Colon Disease

For years, the wider community on the web had agreed to use a colon (:) to separate a name from the identification of the vocabulary it comes from. A colon was a legal name character, and so it was chosen to be backwards compatible, but in some environments could be interpreted in a new way. For instance, xml:lang was an attribute that could be used on any XML-based markup language to identify the (natural) language being used in the contained content.

But for some reason a new separator was developed for HTML5: the hyphen. For instance:

```
<div role="searchbox"
    aria-labelledby="label"
    aria-placeholder="MM-DD-YYYY">03-14-1879</div>
```

apparently re-inventing namespaces.

This also went against another of their design principles: Do not Reinvent the Wheel.

11.4 Reinventing the Wheel

Despite not reinventing things being one of the design principles, nevertheless that precept wasn't followed. As has been noted:

> *"The amount of "not invented here" mentality that [pervades] the modern HTML5 spec is odious. Accessibility in HTML5 isn't being decided by experts. Process, when challenged through W3C guidelines, is defended as being "not like the old ways", in essence slapping the W3C in the face. Ian's made it clear he won't play by the rules. When well-meaning experts carefully announce their opposing positions and desire for some form of closing the gaps, Ian and the inner circle constantly express how they don't understand."* [5]

"Not invented here" (NIH) syndrome is often warned against in design books. For instance:

"Four social dynamics appear to underlie NIH:

- *Belief that internal capabilities are superior to external ones.*
- *Fear of of losing control.*
- *Desire for credit and status.*
- *Significant emotional and financial investment in internal initiatives."* [13]

Many groups had already solved problems that HTML5 should have used, but HTML5 decided to reinvent, usually with worse results, since they were for areas that they were not experts in.

11.5 Not Invented Here: Microdata

To take an example, consider RDFa. This came as the result of the question: *How should you represent general metadata in HTML?*

In 2003 a cross-working-group task force was created of interested parties to address the problem. This produced in 2004 a first working draft of RDFa, which in 2008 finally became the RDFa Recommendation [20], representing more than 5 years of work, consensus, and agreement on how metadata should be represented in HTML and related technologies.

Then a year later in 2009 the HTML5 group created *Microdata* out of the blue [15], with no warning, and no discussion or consultation, clearly copied from RDFa (it used the same attributes), but different, and less capable. This created a lot of confusion in the web community, muddying the Semantic web area. In 2013 Microdata was abandoned, by which time the whole semantic area had been damaged. Microdata has since been periodically revived.

11.6 Forward compatibility: Empty elements

One major improvement that XML introduced was a new notation for empty elements: <br/>. This one simple change meant that you could parse a document without a DTD or Schema; you could parse any document without knowledge of the elements involved, which made the parser forward-compatible. Incomprehensibly, HTML5 dropped the requirement for this notation (probably because of Irritated by Colon Disease), meaning that a processor now has to know which elements are empty, and making it impossible to add new empty elements to HTML (since it would break compatibility).

11.7 Programming

One of the problems of HTML5's dependency on programming to solve their design problems, and using Javascript as the basis of functionality, is that standardisation has become compromised.

As an example take CSS presentation mode which allows you to specify how any document should be formatted when doing a presentation. Alas, HTML5 has taken the approach that you can do this better in Javascript: no browser supports Presentation Mode any more, and consequently numerous Javascript packages have emerged to do presentation instead. But they are all different! This means that you have to choose one of the available packages, and format your slides according to the requirements of that package. However, if that package is no longer supported, or doesn't run on a new browser, or the licence changes and you have to change to another package, you are forced to change all your documents. There is no standardisation.

Effectively, programmers are now doing the document design, so all the documents become proprietary, and there is no interoperability, which is the whole point of standards. This is also why there are so few new elements in HTML5: they haven't done any design, and instead said "if you need anything, you can always do it in Javascript".

11.8 Frameworks

Another aspect of this is that instead of the HTML5 group designing HTML, frameworks have emerged, so that now we have some twenty-odd versions of HTML instead of just the one. To use facilities you need, you have to decide which framework to use, all of which are single-sourced, and different, and hope that it stays alive, remains supported, works on all available browsers, and that they don't change the licencing agreement, because otherwise you are going to have to rewrite your whole website.

> *"What flavor of Javascript are you going to use? Are you gonna use a transpiler? From what language? Grunt? Gulp? Bower? Yeoman? Browserify? Webpack? Babel? Common.js? Amd? Angular? Ember? Linting? What am I talking about? Am I mixing things up? Am I confused?"*
>
> *"Talking to the community about my "analysis paralysis loop" caused by the excessive amount of available tools to choose from and to investigate resulted in the community suggesting to try out, spend time, learn and investigate four more technologies that I haven't even considered in the first place. Good job, Javascript!"* [18]

The use of frameworks has created bloat, slowed the web, and limited accessibility. To look at the web-page of one single tweet of 140 characters, you have to download just under a megabyte. It's 5200 lines of HTML before you even get to the five Javascript packages. The whole of James Joyce's Ulysses is only half as long again.

> *"Because of #GDPR, USA Today decided to run a separate version of their website for EU users, which has all the tracking scripts and ads removed. The site seemed very fast, so I did a performance audit. How fast the internet could be without all the junk! It went from a 5.2MB download to 500KB and a load time of more than 45 seconds to 3 seconds, from 124 (!) JavaScript files to 0, and from a total of more than 500 requests to 34."* [8]
>
> *"Many developers who have grown up only using frameworks have a total lack of understanding about the fundamentals of HTML, such as valid and semantic markup ... This is of great concern as semantic markup is one of the core principles of an accessible web."* [21]

11.9 Complexity

Finally, HTML5 has become so complex, that implementers have found it hard to implement. This has led to an impoverishment of the browser landscape, several browsers, even Microsoft!, having given up trying and instead just put a new wrapper around Google's Chrome browser.

This is regrettable, giving a single player a disproportional power over the web, and risking turning the web into a monoculture.

12 CONCLUSION

A sustainable web needs Modularity, Extensibility, Accessibility, and Standardisation, based on Declarative Principles. A 100 year web is needed because it is the way now that information is distributed. The web pages that are being created now need to be readable in 100 years time, just as 100-year-old books are still readable. Requiring a web-page to depend on a particular 100-year-old implementation of Javascript and a framework which hasn't been supported for 70 years and of which the creators are all dead is not in any sense future-proof.

The web started off as a simple, easy-to-use, easy-to-write-for infrastructure. Programmers, having taken over HTML, have remodelled it in their own image, and made it complicated, hard to implement, and hard to write for, excluding many potential creators.

Hopefully, in the not-too-distant future, the web community can come together again to try and undo the damage being inflicted on the web by the implementors, and bring it back to its declarative roots. At least declarative markup is easier to keep alive because it is independent of implementation!

REFERENCES

[1] James Craig *et al.* (eds), 2014, Accessible Rich Internet Applications (WAI-ARIA) 1.0, W3C, https://www.w3.org/TR/wai-aria-1.0/
[2] Mark Baker *et al.* (eds), 2000, XHTML Basic, W3C, https://www.w3.org/TR/2000/REC-xhtml-basic-20001219/
[3] Stewart Brand, 1993, How Buildings Learn, Viking Press, ISBN 0140139966
[4] H,W, Lie *et al*, (eds), 1996, Cascading Style Sheets, level 1, W3C, https://www.w3.org/TR/REC-CSS1/
[5] Kyle Weems, 2009, Behold Leviathan, Confused, http://cssquirrel.com/blog/2009/08/03/behold-leviathan-confused/
[6] CWI, 2018, CWI celebrates 30 years of Open Internet in Europe, https://www.cwi.nl/news/2018/cwi-celebrates-30-year-of-open-internet-in-europe
[7] Anne van Kesteren *et al.* (eds), 2007, HTML Design Principles, W3C, https://www.w3.org/TR/html-design-principles/
[8] Marcel Freinbichler. 2018, Tweet, https://twitter.com/fr3ino/status/1000166112615714816
[9] History of the World Wide Web, Wikipedia, https://en.wikipedia.org/wiki/History_of_the_World_Wide_Web
[10] Dave Raggett *et al.* (eds), 1997, HTML 4.0 Specification, W3C, https://www.w3.org/TR/REC-html40-971218/
[11] WHATWG, 2022, HTML5, WHATWG, https://html.spec.whatwg.org/multipage/
[12] Wikipedia, JavaScript Style Sheets, https://en.wikipedia.org/wiki/JavaScript_Style_Sheets
[13] William Lidwell *et al.*, 2010, Universal Principles of Design, Rockport Publishers, ISBN 1-59253-587-9
[14] Murray Altheim *et al.* (eds), 2001, Modularization of XHTML, W3C, https://www.w3.org/TR/2001/REC-xhtml-modularization-20010410/
[15] Ian Hickson (ed.), 2010, HTML Microdata, W3C, https://www.w3.org/TR/2010/WD-microdata-20100304/
[16] Norwich Record Office, 2016, The Norwich Computer, 1957, https://norfolkrecordofficeblog.org/2016/04/29/the-norwich-computer-1957/
[17] Steven Pemberton, 2020, On the design of the URL, in Proc. Declarative Amsterdam 2020, Amsterdam, The Netherlands, https://declarative.amsterdam/article?doi=da.2020.pemberton.design
[18] 'pistacchio', 2016, I'm a web developer and I've been stuck with the simplest app for the last 10 days, Medium, https://medium.com/@pistacchio/i-m-a-web-developer-and-i-ve-been-stuck-with-the-simplest-app-for-the-last-10-days-fb5c50917df#.i7o9ivu3x
[19] Melinda Grant *et al.* (eds), 2006, XHTML-Print, W3C, https://www.w3.org/TR/2006/REC-xhtml-print-20060920/
[20] Ben Adida *et al.* (eds)., 2008, RDFa in XHTML: Syntax and Processing, W3C, http://www.w3.org/TR/2008/REC-rdfa-syntax-20081014/
[21] Russ Weakley, 2015, Front End Frameworks - are they accessible? Slideshare, https://www.slideshare.net/maxdesign/front-end-frameworks-are-they-accessible
[22] John M. Boyer (ed.), 2009, XForms 1.1, W3C, https://www.w3.org/TR/xforms11/
[23] Tim Bray *et al.* (eds), 1998, Extensible Markup Language (XML) 1.0 http://www.w3.org/TR/1998/REC-xml-19980210
[24] Steven Pemberton *et al.* (eds), 2000, XHTML™ 1.0: The Extensible HyperText Markup Language, W3C, http://www.w3.org/TR/2000/REC-xhtml1-20000126
[25] Murray Altheim *et al.* (eds), 2001, XHTML™ 1.1 - Module-based XHTML, W3C, https://www.w3.org/TR/2001/REC-xhtml11-20010531/
[26] Mark Birbeck *et al.* (eds.), 2010, XHTML 2.0, W3C, https://www.w3.org/TR/2010/NOTE-xhtml2-20101216/
[27] 石川雅康(ISHIKAWA Masayasu), 2002, An XHTML + MathML + SVG Profile, W3C, https://www.w3.org/TR/XHTMLplusMathMLplusSVG/