Weblint: Just Another Perl Hack

Neil Bowers[1]

Canon Research Centre Europe

Abstract

Weblint is a utility for checking the syntax and style of HTML pages. It was inspired by *lint* [15], which performs a similar function for C and C++ programmers. Weblint does not aspire to be a strict SGML validator, but to provide helpful comments for humans. The importance of quality assurance for web sites is introduced, and one particular area, validation of HTML, is described in more detail. The bulk of the paper is devoted to weblint: what it is, how it is used, and the design and implementation of the current development version.

1. Introduction

Web sites are becoming an increasingly critical part of how many companies do business. For many companies web sites are their business. It is therefore critical that owners of web sites perform regular testing and analysis, to ensure quality of service.

There are many different checks and analyses which you can run on a site. For example, how usable is your site when accessed via a modem? An incomplete list of similar analyses are given at the start of Section 2.

The bulk of this paper is concerned with *weblint*, a tool which concentrates on syntactic and stylistic checking of HTML documents. The second part of Section 2 expands on the reasons why you should consider validating the HTML content of your site.

In section 3 a selection of other quality assurance tools is presented, and related to the different tests described in Section 2. Those presented overlap with weblint in some way. A more complete list of tools can be found on the weblint site, and will be described in a subsequent paper.

Section 4 introduces weblint, and the requirements which have driven its development. The remainder of the section describes the user-visible aspects: how to run weblint, sample output, and how weblint can be configured to personal taste.

Implementation details are presented in Section 5. The general algorithm is described, followed by the software architecture of the current development version.

The conclusion opens with a summary of the information and opinions given in this paper. A selection of the lessons learned over the last four years is given, followed by plans for the future, and related ideas.

2. Web Site Quality Assurance

The following are some of the questions you should be asking yourself if you have a web presence. I have limited the list to those points which are relevant to weblint.

- Is the HTML content of the pages correct with respect to a given HTML DTD? I'll refer to this as *strict HTML validation*.
- Do your pages render as you intended on the browsers which are important to you?
- Do your pages follow your selected style guidelines, or those generally held by the web community?
- Is your web site usable by those with disabilities, or even just people using a text-only browser?
- How usable is your site by people accessing it via a modem?
- Are there any dead links (hyperlinks to pages which have been moved or no longer exist) in your pages? Conversely, are there any pages

[1] `neilb@cre.canon.co.uk`

on your server which aren't referred to (either by your site, or others)?

- How easy is your site to navigate? It is important to remember that users may jump to arbitrary pages on your site, for example as the result of a search.

- Is the appropriate meta information included with all pages? For example, your pages can include `<meta>` tags (not rendered by browsers) which provide the short abstract presented by search engines, and a list of keywords.

- How secure is your web site?

- How well does your site handle different loads through the day?

- Has your site been indexed by the major search engines? Are their records up-to-date?

- Which parts of your site should be disabled for robot access, for example to prevent their inclusion by search engines?

- Is the textual content of your site grammatically correct, and correctly spelled?

- How well do your dynamic pages (for example CGI scripts) handle unexpected input?

This list is by no means complete; a more detailed discussion of this topic is planned for a separate paper.

2.1 Why validate your HTML?

As noted above, weblint is primarily concerned with syntactic and stylistic checking of your HTML documents. This section expands on the reasons why validating your HTML is a worthwhile pursuit.

Humans tend to make mistakes, particularly when performing any repetitive task, and the nature of HTML makes it hard to inspect visually. Furthermore, web page creators often have an incomplete knowledge of HTML, and its correct use. Use of an automated checking tool will catch most of these mistakes. In my experience it also helps users learn more about HTML, and its use.

The only testing many pages get is checking whether they look acceptable in the creator's browser. Browsers are supposed to be lenient in what they accept, doing their best to render even the most mangled HTML. You have no way of knowing how different browsers (and different versions of the same browser) will handle non-conformant HTML. The two most popular browsers do not provide any facilities for checking the correctness of a page. This means that a browser alone is not enough to check your web page.

In addition to humans surfing the web, your pages are also 'viewed' by robots, agents, and other programs. Automated processing of web pages is becoming increasingly important, to handle the volume and complexity of information on the web. Your pages stand a better chance of turning up in search engines if the HTML is well formed, and thus amenable to automatic processing. Following certain guidelines will also help your pages look good when they do turn up in search engines.

More and more HTML is generated from databases and other sources. Automatic generation improves the likelihood of a well-formed site, but validation is still needed to make sure the generators are doing a good job.

If you're not convinced yet, there are a number of articles on this theme [20, 14].

3. Related Work

This section presents a selection of tools which provide analyses for some of the issues described in Section 2. In particular, I have included tools which have functionality which overlaps weblint, or which include weblint.

3.1 Editors and browsers

Amaya is both a browser and authoring tool, being developed by the W3C, as a test-bed for new and emerging standards. Amaya ensures that an HTML page being edited conforms to a specific DTD [25].

The *Arena* browser was W3C's test-bed prior to Amaya. Arena provides feedback on the quality of the web page currently being viewed. Arena is now being developed by Yggdrasil [27].

3.2 SGML parsers and validators

Strict HTML validators are based on an SGML parser, and require a DTD to validate against. Just about every strict validator is based on one of James Clark's parsers, most recently *SP* [10].

Strict validators have the obvious advantage that you are checking against the bible (the DTD). On the down-side, the warning and error messages are usually straight from the parser, and require a grounding in SGML to understand.

Many of the publicly accessible validators also include weblint output in their response, so are really belong in the section on meta tools, described below.

3.3 HTML Checkers

An HTML checker is a tool which performs various analyses of HTML, without attempting to provide strict HTML validation.

Htmlchek is a perl script (also available in awk) which performs syntax checking similar to weblint. It hasn't been updated in the last few years, so doesn't support recent versions of HTML. It still provides useful comments, and includes a number of other useful utilities and documentation [9]. An eerie serendipity: weblint and htmlchek were announced to usenet on the same day!

Bobby identifies problems which may prevent pages from displaying correctly on different browsers. As part of this it will identify changes which would make your pages more accessible to those with disabilities [8]. For example, summary annotations can be added to tables, which is useful for users with speech generating clients

HTML Tidy, by Dave Raggett, is a tool which identifies a number of common HTML errors, and fixes them for you. It can also pretty-print the HTML, and will generate warnings only for problems which it doesn't know how to fix [17].

3.4 Web gateways

There are a number of web-based gateways to the checks and validators listed above. These are usually forms which let you enter a URL or snippet of HTML. URLs are retrieved, the HTML checked, and the results displayed back to the user as HTML. A list of weblint (and other) gateways can be found on the weblint site.

3.5 Broken link robots

One of the most common, and frustrating, problems with web sites is broken links, where the target of a hyperlink no longer exists, or has moved. Broken link robots traverse a web site and check all hyperlinks to see whether the target page exists. At its simplest, this merely consists of sending a HEAD request, and reporting all URLs which result in a 404 response code. Smarter robots will handle redirects (fixing the links) and generate navigational analysis of your site.

The first widely available link validator was *MOMspider* [13], which is still available, but requires an old version of Perl. There are now many such robots available – a more complete list can be found on the weblint site. The number of features and price vary greatly, with no obvious correlation.

There are freely available link validators, such as and *linbot* [29]. These tend to provide just link checking facilities. *SWAN* (SubWeb ANalyzer) performs link checking within a site (and other functions), leaving checking of external links to other programs [30].

Spot is a commercial tool from BT labs which is run on the web site's host machine to analyse a web site for problems. Problems identified include HTML syntax errors, broken links, missing index files, non-portable host references, and summary analyses of your site [7].

3.6 Meta tools

Meta tools incorporate two or more of the categories described above, usually merging the results into a single report. They will often include additional measures, for example by running a spell checker on the text content of a page.

The *WebTechs* service includes strict validation and optional inclusion of weblint output. It can also generate a weight for your web page, including estimated download times for different modem speeds [23].

The Web Consortium runs a meta validator [26] which uses James Clark's SGML parser SP [10], and weblint. This is derived from Gerald Oskoboiny's earlier service, *The Kinder, Gentler Validator* [16].

Web Site Garage offers a number of services, including checking for broken links, performance measures, reducing the size of GIFs (GIF Lube!), and more [3].

3.7 Fixer uppers

Fixer uppers are tools which will correct or improve your web site, usually by modifying the HTML in your pages. *FixIMG* is one of several tools available which will add `WIDTH` and `HEIGHT` attributes to any `IMG` elements which are missing them [2]. *HTML Rename!* renames files and updates hyperlinks to ensure that web pages are portable between Mac, PC, and Unix systems [21]. *HTML Tidy*, mentioned above, can pretty-print your HTML, and fix a number of common errors [17].

Several users have suggested that weblint should optionally fix some of the warnings it identifies. My feeling is that weblint is better kept as a problem identifier, the same way lint doesn't try to fix your source code. Often the HTML is a symptom, and not the problem (e.g. in tools generating HTML).

4. Weblint

In 1994 I started creating web pages, using a text editor and almost no knowledge of HTML (or the web). As I created more pages, I repeatedly made the same basic mistakes. Furthermore, things which worked fine in my version of *Mosaic* had all kinds of weird effects on other clients.

Being a long-time fan of lint and similar tools, I decided that a lint-style HTML syntax checker would be helpful for my colleagues and me. The first version was the result of a few hours' hacking in Perl [22]. Perl was ideally suited to the task, given the rich support for text processing and regular expressions. I tidied it up and released the first public version in October 1994.

This was followed by frequent releases, as users improved my knowledge of HTML, and suggested new warnings. A mailing list of core users, *weblint-victims*, continues to play a major role in the evolution of weblint.

There have been 21 releases of weblint version 1. The releases dates were typically clustered around a new HTML specification, or bursts of enthusiasm. Weblint 2 is now under development, and will shortly be alpha-tested by members of the weblint-victims list.

Weblint is being used on a wide range of platforms, including Unix, DOS, Windows, Amiga, Mac, Archimedes, OS/2, and VMS.

4.1 Weblint requirements and philosophy

There have been a number of requirements and points of philosophy driving the direction and development of weblint. This is not a paper about free software philosophy, but if you're interested in that area, see Eric Raymond's papers on this [18, 19]. The major requirements are:

- Weblint should not impose any specific definition of style, but should be useful for any user, regardless how warped their opinions! As a result, everything in weblint can be turned off. Not all warnings are enabled by default; the warnings enabled by default represent commonly held good practice.

- Weblint isn't the only tool, nor should it be. There are many other tools, some of which are described in Section 2, which can be used in conjunction with weblint to test your web site. One result of this is that the largest part of the weblint web site is the collection of pointers to other tools and resources.

- Weblint should be easy to obtain, install, and use. Use of Perl greatly helped this, as did the number of users who ported weblint to platforms like the Amiga and the Mac (with a GUI), or created web interfaces which let you generate weblint reports for your pages without having to install weblint first.

- The output of weblint should be easy to understand, without requiring intimate knowledge of HTML or SGML.

- Weblint should be flexible in its use, facilitating its invocation from different contexts. It should be easy to run weblint from the command-line, a batch script (for example under *crontab* on Unix), a web page, a robot, or an application.

4.2 Example usage

Consider the following piece of HTML, which we will assume is in a file called `test.html`:

```
<HTML>
<HEAD>
<TITLE>example page
</HEAD>
<BODY BGCOLOR="fffff" TEXT=#00ff00>
   <H1>My Example</H2>
Click <B><A HREF="a.html>here</B></A>
for more details.
</BODY>
</HTML>
```

The following illustrates use of weblint from the Unix command line. The output has been reformatted for readability.

```
% weblint –s test.html
line 1: first element was not
        DOCTYPE specification
line 4: no closing </TITLE> seen
        for <TITLE> on line 3
line 5: value for attribute TEXT
        (#00ffoo) of element BODY
        should be quoted (i.e.
        TEXT="#00ffoo")
line 5: illegal value for BGCOLOR
        attribute of BODY (fffff)
line 6: malformed heading - open
        tag is <H1>, but closing
        is </H2>
line 7: odd number of quotes in
        element <A HREF="a.html>
line 7: </B> on line 7 seems to
        overlap <A>,
        opened on line 7.
```

The DOCTYPE element identifies the definition of HTML which your page uses. The **–s** switch requests short messages, rather than the default traditional lint style of messages:

```
test.html(1): blah blah blah
```

4.3 Categories of output messages

Weblint 1.020 supports 50 different output messages, 42 of which are enabled by default. Most messages are the result of suggestions from users. There's rarely a reason not to add support for a message; if it's useful for one user, it's quite likely to be useful for others. If a message seems esoteric or overly pedantic (I love 'em!), it will be disabled by default. There are three categories of output message:

- *Errors*, which identify things you should fix.
- *Warnings*, which identify things you should think about fixing.
- *Style comments*, which can be configured to match your own guidelines.

Errors are generated for incorrect use of syntax and other serious problems (such as broken links). Some examples:

- Missing close tags for container elements which require the closing tag, such as `<A>`.
- Mis-typed element names, for example `<BLOCKQOUTE>`.
- Forgetting required attributes, such as `ROWS` and `COLS`, for the `TEXTAREA` element.

Warnings are generated for optional syntax which is recommended, potential portability problems, and questionable use of HTML. Some examples:

- HTML allows for attribute values to be quoted using single or double quotes, but many clients and HTML processors can't handle single quotes.
- Weblint can let you know which IMG elements don't have the WIDTH or HEIGHT attributes. Use of these attributes help browsers to layout a page sooner, giving the impression of a faster loader page.
- It is perfectly legal to comment-out markup, but this can be incorrectly parsed by parsers, particularly those of the quick and dirty kind. This is less of a problem than it used to be.
- Use of deprecated markup, such as the `<LISTING>` element, in place of which you should use the `<PRE>` element.

Stylistic comments are used to identify usage which at least one person thinks is questionable. Some examples:

- Use of "here" and other content-free text within anchors (as in "click <u>here</u> to read more about crêpes"). One motivation to fix these is that many search engines will use anchor text [6].
- Use of physical markup (e.g. <B>) rather than logical markup (e.g. <STRONG>).

All output messages have an identifier, which is used when enabling or disabling it. Weblint 2 will let users enable and disable all messages of a given category.

4.4 Configuration

There are three ways to provide configuration information for weblint:

- A site configuration file. This can be useful for defining the style guide for a company or group.
- A user configuration file, `.weblintrc` on Unix systems. An alternate file can be specified. The user's file can either extend or over-ride the site configuration.
- Command-line switches, which over-ride both configuration files.

The most usual configuration is to specify which messages should be enabled, as described above.

4.5 Other ways to run weblint

In addition to the simple usage example above, weblint can be used in a number of different ways.

The **`-R`** switch instructs weblint to recurse in all directories in the local filesystem, so that a set of pages or entire site can be checked with one command. The switch also enables additional warnings, checking whether directories have index files, and reporting orphan pages (which are not referred to by any other page checked).

There are numerous weblint gateways available, which let you use weblint without having to install it. These are CGI forms where you provide the HTML by entering a URL, pasting in the text, or through file upload. If a URL is given, the gateway script retrieves the page, usually using a dedicated retrieval program. The warnings generated by weblint are embedded into the web page generated by the gateway in response to the user's submission.

A robot can be used to invoke weblint on all accessible pages on a site [24]. I have written one, called *poacher*, which is included with the robot module for Perl [5]. Poacher also performs basic link validation.

4.6 Experience with weblint

Weblint 1 is just one large script, almost 2000 lines of Perl 4. The script's monolithic nature made it easy to install, and use of perl 4 made it portable. Both points made weblint harder to maintain.

Increasing diversity in the user base requires weblint to be more configurable and flexible. For example, many editing and generation tools insert tool-specific markup (elements and attributes) in the generated HTML. These result in noise, which hides the useful weblint output.

There are now over 20 public gateways to weblint, and I regularly receive requests for a standard gateway distribution, particularly for installation behind firewalls, e.g. for intranet use.

All of the above points and more prompted the move to weblint 2, currently in development. The next section, on implementation, describes this version.

5. Implementation

This section provides a high-level picture of the implementation of weblint. First the basic algorithm is described, followed by a summary of the software architecture for weblint 2. The most important components of the architecture are summarised, along with the third party code which underpins some components.

All of the weblint scripts and modules are implemented in Perl [22]. All 21 releases of weblint 1 were in Perl 4, since that was widely available on many different platforms. For the last few releases this was at times frustrating, since all my other Perl coding was done with the latest version of Perl 5.

Weblint 2 is a fairly major rewrite, and requires Perl 5.004 or later, which provides numerous improvements over Perl 4 [22]. Two of the most significant additions were support for object-oriented programming, and richer data structures. One of the main goals for weblint 2 was to split the original weblint script into appropriate classes (Perl modules). The distribution now follows CPAN guidelines, which means that installation should consist of these four commands:

```
% perl Makefile.PL
% make
% make test
% make install
```

The build, test harness, and installation support is all provided by the MakeMaker module [28] and friends, which are part of the standard Perl distribution.

5.1 Basic weblint algorithm

Weblint is basically a stack machine with an ad-hoc parser, which uses various heuristics to keep things together as it goes along. The heuristics are based on commonly-made mistakes in HTML. The ad-hoc aspects of weblint are provided in an effort to minimise the number of warning cascades, where a single problem generates a flurry of error messages.

The file being processed is tokenised into start tags (possibly with attributes), text content, and end tags. When an opening tag is seen, it is pushed onto the main stack. Closing tags result in the stack being popped. Certain elements require special processing, such as comments, SCRIPT and STYLE.

A secondary stack comes into play when unexpected things happen, like overlapping elements, as with the example in Section 4.2. The second stack holds unresolved tags, and where they appeared.

For each token type, a number of checks are made. These may involve just the token itself, or its context, which can include the current state of the stack, the secondary stack, and the history of elements seen.

5.2 Software architecture

This section describes the major components of the software architecture for weblint 2.

5.3 Scripts

The *weblint* script is now a wrapper around the modules described below, with documentation for the user who doesn't want to know about the existence of the modules.

The *gateway* is a CGI script which provides a web-based interface for driving the Weblint module. Since there are many weblint gateways existing, the gateway script for weblint 2 is designed to facilitate customisation, modification, and other tinkering.

There are already robots in existence which use weblint [24, 5]. The Weblint module from weblint 2 makes it easier to embed weblint functionality in a robot, such as a link checker. The robot for Canon's public search engine uses weblint to check all of Canon's public web pages [4].

5.4 Weblint module

The weblint module is a Perl class which encapsulates the HTML checking functionality. This makes it easy to embed weblint functionality into any application, where previously weblint would be run in a separate process.

The simplest use of the module is:

```
use Weblint;
$weblint = Weblint->new();
$weblint->check_file($filename);
```

The weblint script is just a fancy version of the above three lines, to support configuration, online help, etc.

In addition to the `check_file` method above, it provides `check_string` and `check_url` methods. The latter requires the LWP modules, described in Section 5.7. If you don't have LWP installed, you can still use weblint, but the `check_url` method won't be available.

5.5 Weblint HTML modules

These modules encapsulate the information which is needed by weblint when checking against a specific version of HTML. By default Weblint will check against HTML 4.0, which is defined in the module `Weblint::HTML40`. Other modules define the non-standard extensions supported by Microsoft (Internet Explorer) and Netscape (Navigator).

This makes it easier to update support for different versions of HTML, and for third parties to provide their own definitions.

The information in an HTML module includes:

- Valid elements, and their content model (are they containers?)
- Valid attributes, and legal values for attributes (expressed as regular expressions)
- Legal context for elements

The HTML modules are basically sets of tables which are used to drive the operation of the Weblint module. At the moment the tables are not generated from DTDs, though this is something I plan to investigate further. Some of the information in the HTML modules cannot be automatically inferred from DTDs, given the sorts of checks which weblint performs.

5.6 Warnings module

All warnings generated by weblint are encapsulated in the `Weblint::Warnings` module. The module provides methods for emitting warnings, as well as configuring the supported set of warnings. The Weblint class uses this by default, but a different class can be used in its place.

The warnings module can be sub-classed, and the new warnings class installed in Weblint. This might change the wording of warnings (e.g. verbose warnings), or change the way warnings are emitted. The gateway script uses a subclass to provide warnings more appropriate to the web page context.

5.7 Other modules

A key tool in the development of weblint has been the test-suite. This serves two purposes: basic testing of the different modules, and a large test set of HTML samples, which are believed to be valid or invalid for specific versions of HTML. The test-suites use support routines from the Weblint::Test module.

The Weblint::Config module encapsulates the configuration parameters for weblint. This is used to handle parsing of command-line parameters, as well as configuration files. Weblint supports a site-wide configuration file, and user-specific configuration.

The Weblint::Constants module defines constants which are used in the different weblint modules.

All retrieving of pages and similar operations are performed using Gisle Aas' excellent LWP package. LWP is a collection of modules and scripts for dealing with HTTP, HTML, and friends [1].

6. Conclusion

Weblint is a useful script for performing syntax and style checking (as opposed to strict validation) on web pages. It can be invoked in several different ways, and the warnings generated configured to personal taste. It has been widely used since 1994, on a surprisingly wide range of platforms.

Weblint does not try to provide a complete solution, but is intended to be used in conjunction with other tools, such as fixer uppers, link analysers, strict validators, logfile analysis tools, etc.

Weblint has evolved according to the *Bazaar* model, as defined by Eric Raymond in his paper *The Cathedral and the Bazaar* [18]. The bazaar model is characterised by groups of individuals working together on a program or system, as typified by the Linux development and evolution. The cathedral model is characterised by development by individual wizards or small groups.

A key element of this model for weblint has been the weblint victims, a list of the most active users. The victims have educated me about HTML, what constitutes a useful tool, and provided the inspiration and source for much within weblint. Various branch versions of weblint have been created by members of this community, with all of them offered for folding back into the standard version. This model is further expounded in Eric's second paper on the open software process [19].

6.1 Future plans and ideas

The following list gives some of the things on my 'todo list' related to weblint. Some of them have been there for a while, and will likely stay there for a while longer yet. Some will be addressed in weblint 2.

- Much greater configurability. For example, to provide additional examples of content-free text, custom elements and attributes, etc.

- Driving weblint with a DTD: generating the HTML modules used by weblint, and test-cases for the test-suite.

- Internationalisation and localisation. Masayasu Ishikawa has done a lot of work in this area, which is being folded into Weblint 2.

- Support for 'plugins' which are used to validate non-HTML content (e.g. to validate stylesheets). This may require an outer framework, where weblint is just one such plugin, for HTML.

- Better tracking of other tools and resources, using a database, submission form, and some kind of robot.

- A style guide for web sites and pages, based on the checks performed by weblint and other tools.

- An article on the complete picture of validating a web site.

- Page-specific configuration of weblint: configuration information embedded in comments, which traditional lint supports [11].

I am always happy to hear suggestions for improving weblint.

7. Getting weblint

You can get the latest version of weblint, and more information, from the weblint home page:

`http://www.cre.canon.co.uk/~neilb/weblint/`

The site also contains links for all of the resources mentioned in this paper:

- online versions of papers.

- the tools and other software described.

8. Acknowledgements

I would like to thank the weblint victims for all their effort on weblint, and patience with me.

I would also like to thank Jane Haslam, Gareth Rees, Ave Wrigley, Martin Portman, and Andy Wardley, who reviewed this paper on extremely short notice, and provided excellent comments.

9. References

[1] Gisle Aas, *LWP – Library for WWW access in Perl*. `http://perl.com/CPAN/modules/by-module/LWP/`

[2] Patrick Atoon, *FixIMG – add WIDTH and HEIGHT to IMG tags*. `http://www.sci.kun.nl/guide/`

[3] AtWeb, *Web Site Garage*. `http://www.websitegarage.com/`

[4] Neil Bowers, *CS-Web: A search engine for Canon's web sites*. `http://csweb.cre.canon.co.uk/`

[5] Neil Bowers, *WWW::Robot – a perl class which implements the traversal engine for a web robot*. Available from the Comprehensive Perl Archive Network: `http://www.perl.com/CPAN/`

[6] Sergey Brin & Lawrence Page, *The anatomy of a large-scale hypertextual Web search engine*. Proceedings of the 7th International World Wide Web Conference.

[7] British Telecom, *Spot – Server Problem Overview Tool*. `http://transend.labs.bt.com/spot/`

[8] Center For Applied Special Technology (CAST), *Bobby*. `http://www.flfsoft.com/bobby/`

[9] Henry Churchyard, *htmlchek*. `http://uts.cc.utexas.edu/~churchh/htmlchek.htm`

[10] James Clark, *SP – An SGML System*. `http://www.jclark.com/sp/`

[11] I. F. Darwin, *Checking C Programs with lint*. O'Reilly & Associates, 1991.

[12] Electronic Software Publishing Corporation, *LinkScan*. `http://www.elsop.com/linkscan/`

[13] Roy T. Fielding, *Maintaining distributed hypertext infostructures: Welcome to MOMspider's Web*. Proceedings of the 1st International World Wide Web Conference.

[14] Mark Gaither, *Why validate your HTML*. `http://www.webtechs.com/html/validate.html`

[15] S. C. Johnson, *lint, a C Program Checker*. Computer Science Technical Report Number 65, 1978.

[16] Gerald Oskoboiny, *A Kinder, Gentler Validator*. `http://ugweb.cs.ualberta.ca/~gerald/validate/`

[17] Dave Raggett, *Clean up your Web pages with HP's HTML Tidy*. Poster, 7th International World Wide Web Conference. `hhtp://w3.org/people/Raggett/tidy`

[18] Eric S. Raymond, *The Cathedral and the Bazaar*. Presented at the first Perl conference, and elsewhere. See `www.opensource.org`

[19] Eric S. Raymond, *Homesteading the Noosphere*. The sequel paper to Cathedral and the Bazaar. See `www.opensource.org`

[20] Tom Sanders, *Why Validate Your HTML* `http://www.earth.com/bad-style/why-validate.htm`

[21] VisionTec Communications, *HTML Rename!* `http://www.visiontec.com/rename/`

[22] Larry Wall, Tom Christiansen, & Randal Schwartz, *Programming Perl*. Published by O'Reilly & Associates, 2nd edition, 1997.

[23] Web Techs, *Web Techs Validation Service*. `http://valsvc.webtechs.com/`

[24] Allison Woodruff, Paul M. Aoki, Eric Brewer, Paul Gauthier, & Lawrence A. Rowe, *An Investigation of Documents from the World Wide Web*. Presented at the 5th International World Wide Web Conference. `http://epoch.cs.berkeley.edu:8000/~woodruff/inktomi/`

[25] W3C, *Amaya – W3C's Browser / Editor*. `http://www.w3.org/Amaya/`

[26] W3C, *W3C Validator*. `http://validator.w3.org/`

[27] Yggdrasil, *The Arena Web Browser*. `http://www.yggdrasil.com/Products/Arena/`

[28] Andy Dougherty, Andreas Koenig & Tim Bunce, *ExtUtils::MakeMaker – create an extension makefile*. Module included in the standard Perl distribution.

[29] Marduk Enterprises, *Linbot*. `http://home1.gte.net/marduk/linbot`

[30] Tom Verhoef, *SWAN – SubWeb Analyser*. `http://www.win.tue.nl/cs/wstomv/swan/swan.html`