Reverse Engineering a CSS Coding Conventions Catalogue

Boryana Goncharenko
Universiteit van Amsterdam, The Netherlands;
Amazon, The Netherlands
boryana.goncharenko@gmail.com

Vadim Zaytsev
Universiteit van Amsterdam, The Netherlands;
Raincode, Belgium
vadim@grammarware.net

Abstract

Coding conventions are preferences of a particular programming style, used to preserve code base consistency and maintain readability. Many practitioners design and publish their custom style guides. In this paper we investigate what conventions exist for the Cascading Style Sheets (CSS) language. We present a categorised catalogue of the discovered CSS style guides with examples for each convention. The catalogue was subsequently used in creating CssCoco [1], a tool for detecting violations based on the specification of conventions.

1. Introduction

Coding conventions is a term comprising a range of rules including indentation, layout, syntactic structure preference and code (anti)patterns. They shape a specific programming style in the context of a software language, a project or an organisation. Code conventions are used to ensure code quality and consistency, which in turn is known to improve the readability and maintainability of the code [30, 31]. We claim it to be an important ingredient in software language usability, since conventions not only demonstrate linguistic idioms, but also classify them into useful and harmful ones, thus providing unprecedented insight and empirically obtainable indication of language features design.

Organisations often design custom style guides that embody conventions used in their projects. This practice is well adopted in the Cascading Style Sheets (CSS) community: W3C has not published a canonical style guide, so many practitioners design their own conventions. Typically, such style guides take the form of natural language descriptions supported by code examples. This is the case with the style guides of Google [5], GitHub [14] and WordPress [2].

PLATEAU '16 1 Nov, 2016, Amsterdam, The Netherlands
Copyright © 2016 held by owner/author(s). Publication rights licensed to ACM.
ACM 978-1-nnnn-nnnn-n/yy/mm. . . $15.00
DOI: http://dx.doi.org/10.1145/nnnnnnn.nnnnnnn

In this paper we investigate which conventions for CSS exist and present a catalogue based one the discovered style guides. We focus on conventions for manually written CSS. Our way to seek conventions and a summary of findings is described in § 2. Related work and terminology are discussed in § 3. In § 4 we present the conventions catalogue. § 5 concludes the paper with a short discussion.

2. Research Method

To discover existing CSS code conventions, two searches with the keywords `CSS code conventions` were made using the search engines DuckDuckGo[1] and Google Search[2]. The first 100 results of each search were analysed, filtering out conventions referring to CSS "preprocessors" like LESS or SASS. In case the result contained links to other style guides, these references were added to the corpus as well.

The process of discovering conventions encountered a number of issues. Specifically, some of the conventions were overgeneralised (did not contain sufficient data to be applicable), incorrectly described (conflicting with the examples), ambiguous (open for interpretation) or underspecified (can only be inferred from their implicit use in other rules) [1].

As a result of the searches, **28** CSS style guides were discovered [2–29]. Authorship of these conventions goes back to CSS professionals, open source communities and commercial companies. The total number of conventions in the discovered style guides is **471**. However, since style guides often share the same conventions and even refer to one another, only **143** of the conventions are unique.

Each convention in the corpus was further categorised based on the type of constraints it specifies:

Layout category (§ 4.1, **35** conventions) contains rules that constrain the overall layout of the code, indentation rules, line splitting, bracket positioning, etc.

Sorting category (§ 4.2, **14** conventions) covers element positioning conventions: where style sheet elements should be put, how they should be grouped, sorted, etc.

[1] `http://duckduckgo.com`
[2] `http://google.com`

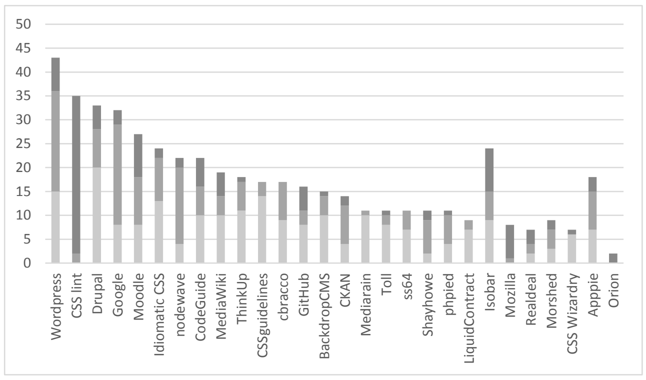

Figure 1. Lighter to darker: layout, syntax, style per guide

Syntax Preference category (§ 4.3, **43** conventions) comprises conventions that express preference of a particular syntax, not aiming at ensuring CSS validity, but affecting the choice among syntactic alternatives: using lowercase or camelCase, single quotes or double quotes, etc.

Programming Style category (§ 4.4, **51** conventions) consists of conventions that put constraints on how CSS constructs are used to achieve feasible goals, they express good and bad practices in the CSS domain and are used to improve maintenance and performance, or to avoid known issues.

Figure 1 presents the distribution of the types of conventions among the discovered style guides.

Most of the conventions appeared in multiple style guides. For example, the guideline *put one space after a colon* is present in 20 out of the 28 style guides. Similarly, 19 of the guides required CSS developers to *use the semicolon* at the end of declarations, even though grammatically such a semicolon is optional.

The catalogue presented in this document was instrumental in creating CssCoco[3], a tool for detecting convention violations based on a specification of a particular selection of conventions chosen to be adhered to. The design of this tool relies on an abstract syntax of a domain-specific language for expressing such conventions, which closely follows the domain modelling ontology, which in turn emerged in analysing the catalogue we present in § 4.

3. Related work

When creating a catalogue of coding conventions, the obvious body of related work to look at, concerns design patterns [32–34]. The most crucial difference is that at the empirical/statistical investigation phase we are not interested in the rationales — if someone wants to detect violations of a particular convention, we must accommodate that desire before asking for their reasons. We are not the first to start cataloguing pattern-like entities for something else than code or architecture — for instance, there was some prior work on metrics [35] and visualisations [36]. Recently Cutumisu et al. have proposed four metrics for assessing quality of such pattern catalogues: usage, coverage, utility and precision [37], but their evaluation procedure relies on substantial existing use of the catalogue, so for us it remains distant future work.

Not all useful catalogues are long lists with unstructured relations like ours: for example, the data structure catalogue by Rüping et al. [38] is a one-page hierarchy graph linking classes (lists, sets, bags, queues, etc). Other algorithmically-driven catalogues seem to fall into the same category, such as the cyber-foraging tactics catalogue [39].

A convention catalogue is only one step away from bad smells catalogues, which are plentiful, covering not just source code, but also software architecture [40], usability traits [41], language design [42], spreadsheets [43], etc. There also exists some (preliminary) work on investigating smells in style sheets [44] with some smells like undoing style receiving more attention [45]. In this aspect our contribution can be seen in broadening the scope and inverting the question: a smell is something already present in software artefacts being discovered; a convention is a set goal for conformance.

In the field of requirements engineering, catalogues are a commonly used methodology, up to the point of having a multi-phase process around their engineering [46,47]: investigation (of state of the art), experimental research (using the catalogue to develop new systems), iterative evolution (of the catalogue following discovered shortcomings and emergent technologies) and collaborative evolution (with refinement based on experiences of versatile clientele). We consider this paper as a comprehensive report on the first stage; CssCoco [1] belongs to the second one, we hope both of them serving as stepping stones to proceed to the later ones.

4. Convention catalogue

This section contains the results of our domain analysis, which is also available in more details online at `http://github.com/boryanagoncharenko/CssCoco/blob/master/analysis.md`. For this paper we have focused mostly on providing an overview and heavily annotating it with illustrative examples.

4.1 *Layout and comments*

Add `/* LTR */` to each use of `left` or `right` [4]
- ✓ `float: right; /* LTR */`
- ✗ `float: right;`

Add `/* LTR */` to each use of `direction:ltr` [4]
- ✓ `direction: ltr; /* LTR */`
- ✗ `direction: ltr;`

Use 4 (or 2) spaces for indentation [2,4–6,9–11,13,14,16–19,21–23]
- ✗ no indentation
- ✗ indentation with tabs
- ✗ indentation with the "wrong" number of spaces
- ? multiple properties in one line

[3] `http://pypi.python.org/pypi?:action=display&name=csscoco`

Indent declarations once [2, 4, 7, 10–12, 15]
- × no indentation
- × some declarations indented twice
- ✓ contents of a `@media` rule can be indented twice
- ✓ comments can have any indentation

Indent all contents of a block [5]
- ✓ everything inside each block is visually indented
- × no indentation
- × inconsistent indentation

Closing curly bracket of a block must be vertically aligned with the selector of the rule [2, 4, 6, 7, 10, 13, 17, 19, 22]
- × inconsistent bracket positioning
- × closing bracket aligned with an opening curly
- × closing bracket aligned with the last property

Vertically align vendor properties [6, 10, 12, 13, 19]

✓
```
.foo {
  -webkit-border-radius: 3px;
  -moz-border-radius:    3px;
  border-radius:         3px;
}
```
×
```
.foo {
  -webkit-border-radius: 3px;
  -moz-border-radius: 3px;
  border-radius: 3px;
}
```

Vertically align values of related properties [12]

✓
```
.bar {
  margin-right: -10px;
  margin-left:  -10px;
  padding-right: -10px;
  padding-left:  10px;
}
```
×
```
.bar {
  margin-right: -10px;
  margin-left: -10px;
  padding-right: -10px;
  padding-left: 10px;
}
```

Indent multiline selectors [25]

✓
```
.class1,
    .class2,
    .class3,
    .class4 { font-size: 80%; }
.otherClass { font-size: 2em; }
```

Rulesets in `@media` should be indented [2, 4, 15]
- × no indentation

Comments should be indented with the thing they describe [4, 6, 15]

✓
```
/* a fix */
color: red !important;
```
×
```
     /* a fix */
color: red !important;
```

Place comments on a new line [7, 22]

✓
```
/* a fix */
color: red !important;
```
× `/* a fix */ color: red !important;`

Add comments after a space after the last `;` [8]

✓
```
height: 100%;
margin-bottom: 1px; /* scrollbars! */
```

Add one blank line between adjacent rulesets [2, 5–7, 10–14, 17, 21]
- × no blank lines between rulesets
- × two or more blank lines

Single-line rules may take adjacent lines [7, 12, 15]
- × multiple single-line rules per line
- ? one line between single-line rulesets
- × two or more lines between rulesets

Values should be on same line as names [7, 12]

✓ `margin: 0px 20px 0px 10px;`

×
```
margin: 0px
        20px
         0px
        10px;
```

Values should not appear on one line [8]

× `margin: 0px 20px 0px 10px;`

✓
```
margin: 0px
        20px
         0px
        10px;
```

No space before colons [4, 10, 15, 17, 18, 27]
- ✓ `color: red;`
- × `color : red;`

Put one space after colons [2, 4–7, 9–12, 14–18, 20–22, 26]
- × `color:  red;`
- ✓ `color: red;`
- × `color:red;`

Put no spaces after colons [27]
- ✓ `color:red;`
- × `color: red;`

Put one space between the last selector and the block [2, 4, 5, 7, 9, 11–14, 16, 17, 19–22]
- × `a{color:blue;}`
- ✓ `a {color:blue;}`
- × `a  {color:blue;}`

Opening brace must be on the same line as the last selector [10, 11, 17, 18, 22, 26]
- ✓ `a{color:blue;}`
- ✓ `a {color:blue;}`

×
```
a
{color:blue;}
```

Put the first declaration on a newline after the opening curly brace [11, 12, 17, 18]
- × `a{color:blue;}`
- × `a {color:blue;}`

✓
```
a
{color:blue;}
```
×
```
a

{color:blue;
}
```

Require newline before a declaration [8]
- × multiple declarations per line

Put exactly one selector per line [2, 4–7, 9, 10, 12–14, 17, 18]
- ✓ `h1 {...}`
- × `h1, h2 {...}`

✓
```
h1,
h2 {...}
```
×
```
h1,

h2 {...}
```

Put every declaration on a new line [2, 4, 6, 7, 9–14, 16, 18, 19, 22]

✓
```
margin-left:10px;
 margin-right:10px;
```
× `margin-left:10px; margin-right:10px;`

Closing brace on a new line [4, 9, 11, 12, 14, 18]

✓
```
a {
color: blue;
}
```
×
```
a {
color: blue;}
```
? `a { color: blue; }`

Short one-liners are allowed [7, 9, 12–14, 25]
- ✓ `a { color: blue; }`

No space after the `(` of functions [2, 4, 15]
- ✓ `rgb(100, 100, 100);`
- × `rgb( 100, 100, 100);`

No space before the `)` of functions [2, 4, 15]
- ✓ `rgb(100, 100, 100);`
- × `rgb(100, 100, 100 );`

Use spaces in CSV lists [2, 4, 7, 9, 11, 15, 19]

✓ `rgb(100, 100, 100);`
✗ `rgb(100,100,100);`

Do not use spaces in CSV lists [9]
✗ `rgb(100, 100, 100);`
✓ `rgb(100,100,100);`

Use newlines to break long values [2, 4]
✓ esp. applicable to `box-shadow`, `text-shadow`, ...

One space separates media feature & value [4]
✗ `@media screen and (min-width:10rem)`
✓ `@media screen and (min-width: 10rem)`
✗ `@media screen and (min-width:  10rem)`

All files should end with a single blank line [4]

4.2 *Positioning and sorting*

Style sheet starts with comment [2, 8, 12, 13, 17, 27]
✓ the first element of the file is a comment

Group related properties [2, 4, 5, 7, 13, 15]
✓
```
.bar {
  margin: 15px;
  padding: 5px;
  color: red;
  font-weight: bolder; }
```
✗
```
.bar {
  margin: 15px;
  color: red;
  padding: 5px;
  font-weight: bolder; }
```

In multiselectors, order the selectors by type: HTML, HTML groupings, classes, IDs [8]
✓ `p, div span, .c, #t {color:red;}`

Order dimensions from outer to inner [8]
✓ `padding:0;width:5px;height:5px;margin:0`
✓ `padding:0;height:5px;width:5px;margin:0`

Order alignments from outer to inner [8]
✓
```
position:absolute; float:none;
overflow:auto; vertical-align:text-top;
text-align:center; direction:ltr;
text-indent:1em; white-space:nowrap;
```

Sort declarations by alphabet [5, 9, 12, 14, 17, 18]
✓ vendor-specific prefixes can be ignored for sorting

Sort declarations by alphabet within group [4]
✓ typical groups: display, positioning, box model, colours, typography, other

Sort vendor-prefixed properties by alphabet [5]
✓ obviously applies when several are present

Order vendor-prefixed values by version [10]
✓
```
/* Safari 4, Chrome 2, iOS 2 */
background-image: -webkit-gradient(...);
/* Safari 5.1+, Chrome 10+, iOS 5 */
background-image:
      -webkit-linear-gradient(...);
```

Use TRBL order for boxing [2, 4, 5, 13]
✓ shorthands of `margin` and `padding` use *top-right-bottom-left* order, and it should be used in freehand declarations as well.

Use TL/TR/BR/BL for corner specifiers [2]
✓ especially applicable to `border-radius*-*`

Keep `@media` rules at the bottom of the sheet [2]
✓ it is also acceptable to not have at-rules

Disallow duplicate properties (rule 1) [48]
✗
```
border: 1px solid black;
border: 1px solid black;
```

Disallow duplicate properties (rule 2) [48]
✗
```
border: 1px solid black;
border: 1px solid red;
```

4.3 *Syntax*

Put a semicolon at the end of each declaration [2, 4–7, 9–13, 15, 18–21, 24, 26]
✓ `.c { color: red;font-size:larger;}`
✗ `.c { color: red;font-size:larger}`

Use `em` instead of `px` [6, 21]
✓ `font-size: 12px;`
✗ `font-size: 1em;`

Use `px` instead of `pt` [8]
✓ `font-size: 12px;`
✗ `font-size: 12pt;`

No `@charset` statements in external sheets [4]
✗ `@charset "UTF-8";`

Omit the protocol from embedded resources [5]
✓ `url(//www.google.com/images/example)`
✗ `url(http://www.google.com/images/example)`

Disallow empty rules [49]
✗ `.myclass { }`
✗ `.myclass { /* Nothing to see here */ }`

Disallow the star hack [50]
✗ `*width: 100px;`

Disallow the underscore hack [50]
✗ `_width: 100px;`

Strings must use double quotes [2, 4, 7, 11, 15, 16]
✓ `font-family: "Arial Black"`
✗ `font-family: 'Arial Black'`

Use single quotes for attrs [2, 4, 5, 7, 9, 15, 19]
✗ `span[class=example] {...}`
✓ `span[class='example'] {...}`
✗ `span[class="example"] {...}`

Use double quotation marks for charsets [5]
✓ `@charset "UTF-8";`
✗ `@charset 'UTF-8';`

Do not use `pt` or `rem` [6]
✗ `font-size: 10pt;`
✓ `font-size: 1em;`
✗ `font-size: 1.5rem;`

Use single quotes in values [5, 11]
✓ `font-family: 'Verdana';`
✗ `font-family: "Verdana";`

Double quote font names with spaces [2, 8]
✗ `font-family: Arial Black;`
✓ `font-family: "Arial Black";`
✗ `font-family: 'Arial Black';`

Do not put quotes in URIs [4, 5, 10, 16]
✓ `@import url(maia.css);`
✗ `@import url('maia.css');`
✗ `@import url("maia.css");`

Use single quotes in URIs [8]
✗ `@import url(vala.css);`
✓ `@import url('vala.css');`
✗ `@import url("vala.css");`

Use hex or `rgba()` for colors [2, 14]
✓ `color: #F00;`
✗ `color: red;`
✓ `color: rgba(50, 100, 150, 1.0);`
✗ `color: rgb(50, 100, 150);`
✗ `color: hsl(10, 100, 250);`
✗ `color: hsla(10, 100, 250, 0.5);`

Use rgba only when opacity is needed [2, 14]
✓ `background-color: rgba(255, 0, 0, 0.5);`
✗ `background-color: rgba(255, 0, 0, 1);`
✓ `background-color: rgb(255, 0, 0);`

Use short hexadecimal values [2, 4, 5, 7, 16, 20]
✓ `color: #E9E9E9;`
✗ `color: #99EE11;`
✓ `color: #111;`
✗ `color: #111111;`

No units for zeros [2, 4–7, 9, 11, 14, 15, 20, 50]
✓ `.box { margin-top: 0; }`
✗ `.box { margin-top: 0px; }`
✗ `.box { margin-top: 0%; }`

Use `px` for `font-size` [14, 23]
✓ `font-size: 12pt;`
× `font-size: 1.5em;`
× `font-size: 2.54rem;`
× `font-size: 3.14cm;`

Line height should not use units [2, 14, 23]
✓ `line-length: 1;`
× `line-length: 1.5pt;`
× `line-length: 0.01cm;`

Use a leading zero for decimal values [2, 5, 9]
✓ `.box { font-size: 0.8em; }`
× `.box { font-size: .8em; }`
✓ `p { text-indent: -0.8em; }`
× `p { text-indent: -.8em; }`

Values in media queries should use `rem` [4]
✓ `@media screen and (min-width: 20rem)`
× `@media screen and (min-width: 450px)`

Presentation words in selectors [19, 21, 23, 26, 28]
× `.blue { ... }`
× `#text-gray { ... }`
× `.light-box { ... }`

Do not abbreviate [6, 21, 28]
✓ `.production { ... }`
× `.prod { ... }`
✓ `#text { ... }`
× `#txt { ... }`

ID and class names must be dash-separated lowercase [2, 5, 6, 9–12, 16, 19, 21, 22, 28]
✓ `.prod { ... }`
× `.Prod { ... }`
✓ `#text-case { ... }`
× `#textCase { ... }`

ID/class names must be lowercase [5, 11, 13, 21, 26]
✓ `.prodrule { ... }`
× `.ProdRule { ... }`
✓ `#textcase { ... }`
× `#text-case { ... }`

Properties should be lowercase [2, 5, 6]
✓ `color: red;`
× `Color: red;`
✓ `margin: 10px;`
× `MARGIN: 10px;`
✓ `-MOZ-border-radius: 5px;`

HTML elements are lowercase [2, 5–7, 11, 13]
✓ `h1 { ... };`
× `H1 { ... };`

Attributes are lowercase [2, 5, 7, 13]
✓ `div[prop] { ... };`
× `div[PROP] { ... };`

Attributes values are lowercase [2, 5, 7, 13]
✓ `img[alt="icon"] { ... };`
× `img[alt="ICON"] { ... };`

Non-string values must be lowercase [2, 5, 7, 13]
✓ `color: red;`
× `color: RED;`
✓ `margin-top: 3px;`
× `margin-top: 3PX;`
✓ `font-family: "Courier New", courier;`

Hexadecimals are lowercase [4, 6, 7, 9, 15, 19, 20]
✓ `color: #fff;`
× `color: #FFF;`

All except comments should be lowercase [8]
✓ `color: #fff;`
× `color: #FFF;`
✓ `.prodrule[attr] { ... }`
× `.ProdRule[ATTR] { ... }`
✓ `color: red;`
× `color: RED;`
✓ `font-family: "courier new", courier;`
× `font-family: "Courier New", courier;`

ID/class names must be camelCase [11, 23, 25]
✓ `.prodRule { ... }`
× `.Prod-rule { ... }`
✓ `#textCase { ... }`
× `#TEXTCASE { ... }`

Avoid using attributes in selector names [25]
× `.red { color: red; }`
× `.center { text-align: center; }`

Forbid specific words in ID and class names [25]
× `.box-top { color: red; }`
× `.head-menu { color: red; }`
× `.left-notice { color: red; }`

IDs & classes should be shorter than 20 chars [8]
× `.table-of-contents-and-figures {...}`

Almost all colors should be in hex [8]
✓ `color: white;`
✓ `color: black;`
✓ `color: transparent;`
× `color: red;`
✓ `color: #ffdec9;`

A class and an ID cannot share a name [8]
× no document should contain both `.x` and `#x`

Multiselectors must have 4− selectors [10]
✓ `h1 { color:red;}`
✓ `h1,h2 { color:red;}`
✓ `h1,h2,h3 { color:red;}`
✓ `h1,h2,h3,h4 { color:red;}`
× `h1,h2,h3,h4,h5 { color:red;}`
× `h1,h2,h3,h4,h5,h6 { color:red;}`

Selectors must have 4− simple selectors [10]
✓ `h1 { color:red;}`
✓ `div h1 { color:red;}`
✓ `div.main h1 { color:red;}`
✓ `div.main h1.cool { color:red;}`
× `div.main[style] h1.cool { color:red;}`
× `div.main[style] h1.cool #a { color:red;}`

4.4 *Style*

Avoid using z-indexes when possible [10]
× `z-index: 100;`

Avoid using `!important` [10, 18]
× `color: red !important;`

Do not use ID selectors [3, 6, 25, 27, 29]
× `#header a {...}`

Disallow `@import` [9, 23, 25, 51]
× `@import url(foo.css);`

Zero or one ID selectors per rule declaration [14]
× `#header .search #quicksearch { ... }`
? `#header, #footer { ... }`

Use px units as a safe fallback for rem units [4]
✓ `font-size: 24px; font-size: 1.5rem;`
× `font-size: 1.5rem;`

Beware of box model size (rule 1) [3]
✓ `width: 100px; border: 1px; box-sizing: border-box;`
× `width: 100px; border: 1px;`
× `width: 100px; padding: 1px;`

Beware of box model size (rule 2) [3]
✓ `height: 100px; border: 1px; box-sizing: border-box;`
× `height: 100px; border: 1px;`
× `height: 100px; padding: 1px;`

Require properties for display (rule 1) [52]
× `display:inline; width:25px;`
× `display:inline; height:25px;`
× `display:inline; margin:10px;`

✗ `display:inline; float:left;`

Require properties for display (rule 2) [52]
✗ `display:inline-block; float:left;`

Require properties for display (rule 3) [52]
✗ `display:block;vertical-align:text-top;`

Require properties for display (rule 4) [52]
✗ `display: table-cell; margin: 10px;`

Disallow adjoining classes [50]
✓ `.foo .bar {color: red;}`
✗ `.foo.bar {color: red;}`

Disallow the `box-sizing` property [50]
✗ `.mybox {box-sizing: border-box;}`

Require compatible vendor prefixes [4, 6, 7, 50]
```
✓ -moz-transform: translate(50px, 100px);
  -o-transform: translate(50px, 100px);
  -ms-transform: translate(50px, 100px);
  transform: translate(50px, 100px);
✗ transform: translate(50px, 100px);
```

Disallow negative indent [50]
✓ `direction:rtl; text-indent:-9em;`
✗ `text-indent:-9em;`

Require standard property (no prefix) [6, 50]
```
✓ -moz-transform: translate(50px, 100px);
  -o-transform: translate(50px, 100px);
  -ms-transform: translate(50px, 100px);
  transform: translate(50px, 100px);
✗ -moz-transform: translate(50px, 100px);
```

Require a fallback color [6, 50]
✓ `color:gray; color:rgba(100,200,100,0.5);`
✗ `color:rgba(100,200,100,0.5);`

Bulletproof font face [23, 50]
✗ `@font-face {src: url('webfont.eot');}`

Do not use too many web fonts [50]
✗ more than five `@font-face` declarations

Disallow regex-looking selectors [9, 50]
```
✓ .mybox[class]{ color: red; }
✓ .mybox[class=xxx]{ color: red; }
✗ .mybox[class~=xxx]{ color: red; }
✗ .mybox[class^=xxx]{ color: red; }
✗ .mybox[class|=xxx]{ color: red; }
✗ .mybox[class$=xxx]{ color: red; }
✗ .mybox[class*=xxx]{ color: red; }
```

Disallow the universal selector [23, 50]
```
✗ * { color: red; }
✓ .selected * a { color: red; }
✗ .selected * { color: red; }
```

Disallow unqualified attribute selectors [50]
```
✗ .myclass[type=text] { color: red; }
✓ .selected [type=text] a { color: red; }
✗ .selected [type=text] { color: red; }
```

Disallow overqualification [2, 9, 16, 23, 24, 29, 50]
✓ `p.active {...} li.active {...}`
✗ `li.active {...}`

No combos of ID/classes with types [2, 5, 6, 24]
```
✓ .error {}
✗ div.error {}
✓ #example {}
✗ ul#example {}
```

Disallow duplicate background images [50]
```
✗ .heart {background: url(sprite.png)
    0 0 no-repeat;}
  .task {background: url(sprite.png)
    0 0 no-repeat;}
```

Disallow too many floats [50]
✗ more than 10 declarations with `float` property

Do not use too many font size declarations [50]
✗ more than 10 declarations of `font-size`

Disallow `outline:none` (rule 1) [50]
```
✓ a:focus { outline:none; }
✗ a { outline:none; }
✓ a:focus { outline: 0; }
✗ a { outline: 0; }
```
[4]

Disallow `outline:none` (rule 2) [50]
```
✓ a:focus { outline:none; }
✗ a { outline:none; }
✓ a:focus { outline: 0; }
✗ a { outline: 0; }
✗ a:focus p { outline: 0; }
✗ a:focus, p { outline: 0; }
```

Disallow qualified headings [50]
✓ `h3 {font-weight: normal;}`
✗ `.box h3 {font-weight: normal;}`

Headings should only be defined once [50]
```
✓ h3 {font-weight: normal;}
✗ h3 {font-weight: normal;}
  .box h3 {font-weight: bold;}
✓ h3 {font-weight: normal;}
  h3:hover {font-weight: bold;}
```

Require fallback on gradient backgrounds [10]
```
✓ background-color:#555;
  background-image:
     linear-gradient(top,#111,#999);
✗ background-image:
     linear-gradient(top,#111,#999);
```

Use the most specific category possible [9, 24]
✗ `.item[folder] > treerow > treecell {...}`

Avoid the descendant selector [2, 24]
✓ `treehead > treerow > treecell {...}`
✗ `treehead treerow treecell {...}`

No child selector with tag category rules [24]
✗ `treehead > treerow > treecell {...}`

Question all usages of the child selector [2, 24]
✗ `div > .sect {...}`

Avoid vendor-specificity unless necessary [24]
✗ `-webkit-border-radius: 1px;`
✗ `-moz-border-radius: 10px;`

No shorthand properties but `border` [8, 9, 21, 22]
```
✓ border: 1px solid black;
✗ background: red url(bg.jpg) no-repeat;
✗ font: 15px arial, sans-serif;
✗ list-style: square outside;
✗ margin: 2cm 3cm 4cm 5cm;
✗ padding: 2cm 3cm 4cm 5cm;
✗ transition: transition: width 2s;
```

No shorthand properties except `background` [8]
```
✗ border: 1px solid black;
✓ background: red url(bg.jpg) no-repeat;
✗ font: 15px arial, sans-serif;
✗ list-style: square outside;
✗ margin: 2cm 3cm 4cm 5cm;
✗ padding: 2cm 3cm 4cm 5cm;
✗ transition: transition: width 2s;
```

No shorthand properties except `list-style` [8]
```
✗ border: 1px solid black;
✗ background: red url(bg.jpg) no-repeat;
✗ font: 15px arial, sans-serif;
✓ list-style: square url(sq.png);
✗ margin: 2cm 3cm 4cm 5cm;
✗ padding: 2cm 3cm 4cm 5cm;
✗ transition: transition: width 2s;
```

Use shorthand `margin` [2, 5, 6, 16, 20, 50]

[4] Note that the implementation provided by CSS Lint takes into consideration the presence of the `:focus` pseudo selector anywhere in the selector. Thus, a rule `a:focus p { outline:0; }` does not yield a warning, and neither does `a:focus, p { outline:0; }`, which is obviously wrong, hence the following rule.

✓ `margin: 10px 15px 25px 15px;`
✗ `margin-top: 10px; margin-bottom: 25px; margin-left: 15px; margin-right: 15px;`

Use shorthand `padding` [2,5,6,16,20,50]
✓ `padding: 1px 2px 3px 5px;`
✗ `padding-top: 1px; padding-bottom: 3px; padding-left: 2px; padding-right: 5px;`

Use shorthand `border` [2,5,6,16,20]
✓ `border: 1px 2px 3px 5px solid black;`
✗ `border-top: 1px; border-bottom: 3px; border-left: 2px; border-right: 5px; border-color:red; border-style:solid;`

Use the shorthand `font` property [2,5,6,16,20]
✓ `.x {font: Fantasy 9em;}`
✗ `.x{font-size:9em; font-family:Fantasy;}`

Use the `list-style` shorthand [2,5,6,16,20]
✓ `ul { list-style: square inside; }`
✗ `ul { list-style-type: square; list-style-position: inside; }`

No `js-` prefixed names in CSS files [9,14]
✓ The `js-` prefix is exclusively for JS files
✓ Use the `is-` prefix for CSS/JS shared state rules

Forbid using `border` to set the border color [8]
✗ `border: 5px solid red;`

Forbid using `background` to set the color [8]
✗ `background: red url(bg.jpg) no-repeat;`

Always set full border value [8]
✓ `border: 1px solid black;`
✗ `border: 1px;`

Shorthands are for when all sides are equal [8]
✓ `padding: 0px;`
✗ `padding: 1px 0px 2px 0px;`
? `padding: 2px 5px;`

5. Conclusion

Practitioners often design and publish custom CSS code conventions that they use in a specific context. In this paper we have found what code conventions for CSS exist and presented a catalogue of the discovered style guides. To find existing code conventions, we used two search engines and analysed the first 100 results of each search. As a result, we discovered 28 style guides containing 471 conventions of which 143 conventions are unique — these were presented in § 4 in the form of positive and negative examples as well as sources where a detailed description and rationale can be found. We also maintain an online version of the catalogue, where each entry is defined and explained, for example:

Description: Disallow `@import`.
Sources: CSS Lint [51], Real Deal [25], Isobar [23], Code Guide [9].
Violations: For performance reasons, the usage of `@import` should be avoided. The following pattern is considered a violation: `@import url(foo.css);`
Actions: Find usage of `@import` statements

Our process of eliminating redundancies was very straightforward and does not merit any specific description like done in more elaborate projects [46] — this simplicity was mainly due to the simplicity of CSS. We have constructed the catalogue of CSS coding conventions as a part of domain analysis. Namely, it was a great help in formulating the ontology of the domain of detecting violations of CSS coding conventions. We have evaluated the result by developing a tool for detecting such violations [1]. However, we claim that the catalogue is a useful artefact that enables additional research opportunities. In the future, by combining it with mainstream repository mining techniques, we can answer questions like the following ones:

◇ How often are such conventions adhered to?
◇ How soon after the start of the project are they typically defined?
◇ Do convention respecting developers get fewer merge conflicts than *unconventional* ones?
◇ What are the sentiments of developers towards the conventions?
◇ Are changes in the base language desirable to build the conventions in?
◇ Can bulk automation of refactorings that enforce adherence to conventions make a useful and appreciated addition to the web developing ecosystem?

We hope to have given some ground on which research questions like these can base their answers. The next steps, as described earlier, should involve incremental improvements of the catalogue towards completeness and coverage of the domain based on creating new tools with it and performing appropriate experiments.

References

[1] Boryana Goncharenko and Vadim Zaytsev. Language Design and Implementation for the Domain of Coding Conventions. Submitted to the 9th International Conference on Software Language Engineering (SLE), pending notification, 2016.

[2] Wordpress. CSS Coding Standards.

[3] Nicholas C. Zakas. Disallow IDs in selectors.

[4] Drupal. CSS coding standards.

[5] Elliot Glaysher. HTML/CSS Style Guide.

[6] Moodle. CSS coding style.

[7] Nicolas Gallagher. Principles of writing consistent, idiomatic CSS.

[8] Michel Bagnol. CSS Coding Style Conventions, 2009.

[9] Mark Otto. Code Guide.

[10] MediaWiki. Manual:Coding conventions/CSS.

[11] John Catterfeld. Code Style Guide: CSS.

[12] Harry Roberts. css {guide:lines;}.

[13] Chris Bracco. CSS Conventions, 2015.

[14] CSS team at GitHub. Code Guidelines.

[15] Backdrop CMS API. CSS Coding Standards.

[16] CKAN. CSS coding standards, 2013.

[17] Tyler Nielsen. Coding standards: HTML/CSS, 2013.

[18] Benjamin Toll. Code Conventions for Cascading Style Sheets.

[19] Simon Sheppard. CSS Naming convention.

[20] Shay Howe. Writing Your Best Code, 2015.

[21] Stoyan Stefanov. CSS coding conventions, 2005.

[22] AlexMA. CSS Coding Conventions.

[23] Isobar. Front-end Code Standards.

[24] David Hyatt. Guidelines for efficient CSS, 2000.

[25] Realdeal. CSS Naming Conventions and Coding Style, 2008.

[26] Morshed Alam. CSS coding guidelines/conventions, 2010.

[27] Harry Roberts. My HTML/CSS coding style.

[28] Apppie. Naming Convention.

[29] Mark Macdonald et al. Orion Coding conventions: CSS, 2014.

[30] Raymond P. L. Buse and Westley R. Weimer. Learning a Metric for Code Readability. *IEEE Transactions on Software Engineering*, 36(4):546–558, July 2010. doi:10.1109/TSE.2009.70.

[31] Taek Lee, Jung Been Lee, and Hoh Peter In. A study of different coding styles affecting code readability. *International Journal of Software Engineering and Its Applications*, 7(5):413–422, 2013.

[32] Walter F. Tichy. A Catalogue of General-Purpose Software Design Patterns. In *TOOLS USA*, pages 330–339. IEEE CS, 1997. doi:10.1109/TOOLS.1997.654742.

[33] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[34] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.

[35] Xavier Franch and Gemma Grau. Towards a Catalogue of Patterns for Defining Metrics over i*Models. In *CAiSE*, volume 5074 of *LNCS*, pages 197–212. Springer, 2008. doi:10.1007/978-3-540-69534-9_16.

[36] Chris Parnin, Carsten Görg, and Ogechi Nnadi. A Catalogue of Lightweight Visualizations to Support Code Smell Inspection. In *SOFTVIS*, pages 77–86. ACM, 2008. doi:10.1145/1409720.1409733.

[37] Maria Cutumisu, Curtis Onuczko, Duane Szafron, Jonathan Schaeffer, Matthew McNaughton, Thomas Roy, Jeff Siegel, and Mike Carbonaro. Evaluating Pattern Catalogs: The Computer Games Experience. In *Proceedings of the 28th International Conference on Software Engineering*, pages 132–141. ACM, 2006. doi:10.1145/1134305.

[38] Andreas Rüping, Franz Weber, and Walter Zimmer. Demonstrating Coherent Design: A Data Structure Catalogue. In *TOOLS USA*, pages 363–375. Prentice Hall, 1993.

[39] Grace A. Lewis and Patricia Lago. A Catalog of Architectural Tactics for Cyber-Foraging. In *Proceedings of the 11th International Conference on Quality of Software Architectures (QoSA)*, pages 53–62. ACM, 2015. doi:10.1145/2737182.2737188.

[40] Joshua Garcia, Daniel Popescu, George Edwards, and Nenad Medvidović. Toward a Catalogue of Architectural Bad Smells. In *Proceedings of the Fifth International Conference on Quality of Software Architectures: Architectures for Adaptive Software Systems (QoSA)*, volume 5581 of *LNCS*, pages 146–162. Springer, 2009. doi:10.1007/978-3-642-02351-4_10.

[41] Diogo Almeida, José Creissac Campos, João Saraiva, and João Carlos Silva. Towards a Catalog of Usability Smells. In *SAC*, pages 175–181. ACM, 2015. doi:10.1145/2695664.2695670.

[42] Soroosh Nalchigar, Rick Salay, and Marsha Chechik. Towards a Catalog of Non-Functional Requirements in Model Transformation Languages. In *AMT*, volume 1077 of *CEUR*, pages 72–81. CEUR-WS.org, 2013.

[43] Rui Abreu, Jácome Cunha, João Paulo Fernandes, Pedro Martins, Alexandre Perez, and João Saraiva. Smelling Faults in Spreadsheets. In *Proceedings of the 30th International Conference on Software Maintenance and Evolution*, pages 111–120. IEEE, 2014. doi:10.1109/ICSME.2014.33.

[44] Golnaz Gharachorlu. Code smells in cascading style sheets: An empirical study and a predictive model. Master's thesis, University of British Columbia, Canada, 2014.

[45] Leonard Punt, Sjoerd Visscher, and Vadim Zaytsev. The A?B*A Pattern: Undoing Style in CSS and Refactoring Opportunities it Presents. In *32nd International Conference on Software Maintenance and Evolution (ICSME)*, 2016. In print.

[46] Samuel Renault, Oscar Mendez-Bonilla, Xavier Franch, and Carme Quer. A Pattern-based Method for Building Requirements Documents in Call-for-tender Processes. *IJCSA*, 6(5):175–202, 2009.

[47] Milene Serrano and Maurício Serrano. Ubiquitous, Pervasive and Mobile Computing: A Reusable-Models-based Non-Functional Catalogue. In *Proceedings of Requirements Engineering@Brazil*, volume 1005 of *CEUR*. CEUR-WS.org, 2013.

[48] Nicholas C. Zakas. Disallow duplicate properties.

[49] Nicholas C. Zakas. Disallow empty rules.

[50] Nicholas C. Zakas. CSS Lint Rules.

[51] Nicholas C. Zakas. Disallow import.

[52] Nicholas C. Zakas. Require properties appropriate for display.

[53] Boryana Goncharenko. Detecting Violations of CSS Code Conventions. In *Pre-proceedings of the Eighth Seminar on Advanced Techniques and Tools for Software Evolution (SATToSE)*, pages 89–91, 2015.

Acknowledgement

The authors express their gratitude to the organisers and reviewers of the Eighth Seminar on Advanced Techniques and Tools for Software Evolution (SATToSE) where the very early state of this project was first presented [53], and to the University of Amsterdam that sponsored our participation.