Code Smells in Cascading Style Sheets: An Empirical Study and a Predictive Model

by

Golnaz Gharachorlu

B.Sc., The University of Tehran, 2008

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

The Faculty of Graduate and Postdoctoral Studies

(Electrical and Computer Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

December 2014

Abstract

Cascading Style Sheets (CSS) is widely used in today's web applications to separate presentation semantics from HTML content. Despite the simple syntax of CSS, the language has some characteristics, such as inheritance, cascading and specificity, which make authoring and maintaining CSS a challenging task. In this thesis, we describe a set of 26 CSS smells and errors, collected from various development resources and propose an automated technique to detect them. Additionally, we conduct a large empirical study on 500 websites, 5060 CSS files in total which consist of more than 10 million lines of CSS code, to investigate which smells and errors are more prevalent and to what extent they occur in CSS code of today's web applications. Finally, we propose a model based on the findings of our empirical study that is capable of predicting the total number of CSS code smells in any given website which can be used by developers as a CSS code quality guidance. A study of unused CSS code on 187 websites and its results are also described in this thesis.

Preface

This thesis presents a large empirical study of code smells in Cascading Style Sheets (CSS) conducted by the author under the supervision of Dr. Ali Mesbah. I was responsible for designing the smell detection algorithms, implementation and evaluation of the tool CSSNose that is an extended version of an open source tool called Cilla [1], collecting data, analyzing results and proposing the predictive model.

The results of this work is under review as a submission to a software engineering journal.

Table of Contents

Appendices

List of Tables

List of Figures

Acknowledgements

I would like to thank Dr. Ali Mesbah for giving me the opportunity to be a member of SALT lab. I appreciate his wise supervision, broad knowledge and careful guidance during my research work.

I would like to acknowledge Dr. Karthik Pattabiraman and Dr. Konstantin Beznosov for the generosity with their time and kindly agreeing to be on my defence committee.

A special thank you to all my friends for being supportive during the time I have been away from my family.

Last but not least, I would like to thank my parents for their dedication and support during all years of my life and education. With no doubt, it was not possible for me to achieve any of my goals without their generosity and kindness.

Dedication

To my parents

Chapter 1

Introduction

Cascading Style Sheets (CSS)[1] is a language for describing presentation semantics in web applications. Along with HTML and JavaScript, CSS is widely used by developers to achieve separation of concerns between styling and structural code. According to recent surveys [2], more than 90% of today's websites use CSS. Despite its wide-spread adoption, CSS has not received much attention from the research community.

Every cascading style sheet contains a sequence of independent rules. Each rule defines a set of styling properties, which gives CSS a relatively simple syntax. Behind this apparent syntactical simplicity, the CSS language features intricate characteristics, such as inheritance, cascading, and specificity, which make authoring and maintaining CSS code a cumbersome and time consuming task for developers [1, 3–5].

As a result, it is common for CSS code to contain smells. Code smells are inappropriate patterns of the code that reflect weaknesses in the design and may cause issues in program comprehension and maintenance in the long term. In a recent study, Mesbah and Mirshokraie [1] conducted an analysis of unused CSS code and found that on average 60% of CSS selectors is unmatched or ineffective in deployed web applications. More recently, Mazinanian et al. [6] studied duplicated code in CSS and found that the extent of duplication in CSS code ranges between 40–90% for the vast majority of the examined CSS files. These studies, however, consider only two types of smells in CSS, i.e., unused and duplicated code.

Our work in this thesis is motivated by the fact that there is currently a lack of a comprehensive empirical study to examine the prevalence and extent of different types of CSS code smells in today's web applications. To the best of our knowledge, this thesis presents the first large-scale empirical study to characterize CSS code smells in web applications.

In this work, we first propose a set of eight new code smells, which are not detected by current CSS analysis tools such as CSS Lint[2] or W3C CSS

[1] `http://www.w3.org/Style/CSS/`

[2] `http://csslint.net/`

VALIDATOR.[3] We implement a CSS smell detector, called CSSNOSE, which detects the proposed CSS smells.

We conduct a large empirical study to examine the prevalence and extent of CSS code issues in today's websites. We investigate correlations between CSS code characteristics and the number of CSS smell types and instances. Our study is conducted on 500 websites consisting of 5,060 CSS files and 10 million lines of CSS code. The analysis includes 26 CSS smell types in total, i.e., eight proposed smells detected by CSSNOSE, 16 existing types of CSS smells detected by CSS LINT, and two validation errors detected by W3C CSS VALIDATOR. In addition, we analyze the extent of the unused CSS code in 187 of these 500 websites.

Our results indicate that among the detected code smells, CSS properties with hard-coded values and undoing style code are the most prevalent code smells occurring in 96% of the 500 websites studied, followed by too long rules and using IDs in CSS selectors included in CSS code of 94% and 90% of the websites, respectively. In terms of extent, on average 36% of properties are hard-coded, 12% of properties apply undoing style, 22% of CSS selectors use IDs, and 12% of rules are too long.

In addition, using the data obtained from our empirical study, we propose a predictive model, capable of predicting the CSS code quality of a given website. Our predictive model shows a high level of accuracy by exhibiting normalized prediction errors as well as residual medians close to zero.

The rest of this thesis is organized as follows. The next chapter describes the CSS language by giving an overview of the main features. We discuss our proposed set of CSS code smells, our detection criteria and mechanism, implementation of our tool and its evaluation in chapter 3. In chapter 4 we present the design of our empirical study followed by the results of the study. Furthermore, an analysis of CSS unused code is included in this chapter. Our CSS predictive model is proposed and evaluated in chapter 5. The thesis closes with discussions, related work and conclusions.

[3] `http://jigsaw.w3.org/css-validator/`

Chapter 2

Background: the CSS Language

In this chapter, we describe the main features of the CSS language that are needed to understand the rest of the thesis.

2.1 CSS Rules

CSS is a language for prescribing the presentation semantics of HTML elements. Cascading style sheets are written as a set of *rules*. Figure 2.1 depicts the grammar of CSS rules. Each rule is composed of a *selector* part

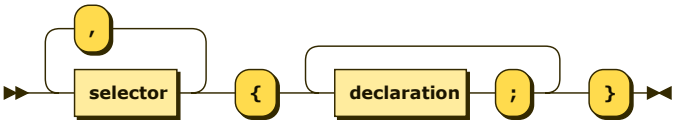


Figure 2.1: CSS Rule Grammar

and a *declaration* part surrounded by curly braces. The selector part selects one or more Document Object Model (DOM) elements from the webpage. The declaration part contains one or more declarations, separated by semicolons. Each declaration includes a styling property name and its value, to be applied to the DOM elements selected by the selector part.

CSS Selectors. A CSS selector can select one or more DOM elements from the webpage. There are three main mechanisms to select DOM elements by the selector type:

- **Element Selectors** select DOM elements based on their element types (e.g., P, DIV, A, SPAN);
- **ID Selectors** are defined using '#' as the prefix followed by an ID name and select DOM elements based on the value of their ID attributes;

- **Class Selectors** are specified by the prefix '.' followed by the class name and select DOM elements which have a class attribute matching the class name of the selector.

In Figure 2.2, three CSS rules are defined based on these mechanisms. The rules have different selector types but they all select the same DOM element, i.e, `p`, to change its colour.

CSS code	DOM

`p { /* element selector */`
`  color: blue;`
`}`
 	
`#ex22 { /* ID selector */`
`  color: blue;`
`}`
 	
`.example { /* class selector */`
`  color: blue;`
`}`
	`<p id="ex22" class="example"> content </p>`

Figure 2.2: Three CSS rules with different selector types selecting the same DOM element

The selector part of the rule can either consist of a simple selector of any of the above three types or a complex selector that combines a number of simple selectors.

Combinators. There are four types of combinators for constructing complex selectors. Assuming that ss_1 and ss_2 are two simple selectors, we can combine them to create a complex selector as follows:

- **Descendant Combinator** (ss_1 ss_2) selects all elements selected by ss_2 that are descendants of elements selected by ss_1;
- **Child Combinator** ($ss_1 > ss_2$) selects all elements selected by ss_2 that are direct children of elements selected by ss_1;
- **General Sibling Combinator** ($ss_1 \sim ss_2$) selects all elements selected by ss_2 that have an element selected by ss_1 as a sibling;
- **Adjacent Sibling Combinator** ($ss_1 + ss_2$) selects all elements selected by ss_2 that have an element selected by ss_1 as a direct sibling;

Selector Unit. In the rest of this work, we refer to each simple selector used in a complex selector as a selector unit. By this definition, we have three types of selector units, namely, ID unit, class unit, and element unit. Figure 2.3 depicts an example of a complex selector composed of one ID

(`#news`), two class (`.headlines` and `.first`), and four element selector units (`ul`, `li`, `span`, and `a`). This complex selector selects all `a` elements that fall under `span` elements, that fall under `li` elements with class `"first"`, under `ul`, under elements with class `"headlines"`, under elements with ID `"news"`.

```
#news .headlines ul li.first span a {
  margin: 10%;
  color: red;
}
```

Figure 2.3: Selector Units

Selectors can also be defined as *pseudo classes* and *pseudo elements*. *Pseudo classes* define styling for a special state of an element. For example, `a:visited` selects only the links visited by the user and not all the links. *Pseudo elements* limit the scope of applying style declarations to specific parts of an element. For example, `p::first-letter` formats only the first letter of the text in `p` elements.

Different selector parts with the same declaration can be *grouped* together to avoid code duplication in CSS. In grouping, selector parts are separated by commas.

2.2 Inheritance and Cascading Order

The concept of inheritance in CSS is based on the fact that child DOM elements inherit styling properties from their parents automatically.

If multiple rules select the same set of DOM elements, then they compete for applying their styling properties to the selected elements. In this case, the cascading order decides which rules are applied ultimately based on the rules' location and specificity.

Location. CSS code can be defined in different positions in a web application:

- Inline CSS Code: It is defined for specific HTML elements using the style attribute;
- Internal or Embedded CSS Code: This type of CSS is located inside a <style> element, added to the HTML document's header;

- External File: In this case, CSS code is authored in external style sheets which are separate files usually with .css extension and are referenced within the HTML document;

For competing rules, the further a rule is from a selected element, the less priority it receives to be applied to that element.

Specificity. The specificity value (SV) of a selector S is a four-digit number, computed as follows:

$$SV(S) = concatenate(a, b, c, d) \tag{2.1}$$

where:

- $a = \begin{cases} 1 & \text{If CSS is inline} \\ 0 & \text{Otherwise} \end{cases}$
- b is total number of ID units in the selector
- c is total number of class units in the selector
- d is total number of element units in the selector

If two or more selectors target the same DOM element to apply their property values, the selector with the larger specificity value receives a higher priority. In case of equal specificity numbers, the selector location will be the determining factor, i.e., the one closest to the DOM element will be selected. By this definition, inline styles have the highest priority. Then there are rules in embedded style sheets which take precedence over rules in external style sheets.

The CSS language also provides the possibility of using *!important* in the declaration part of each rule. A property designated as *important* has the highest priority regardless of its specificity value or location.

Chapter 3

Proposed CSS Smells

In this chapter, we propose a set of eight CSS code smells, collected by analyzing various CSS development resources[4] [5] [6] [7]. To the best of our knowledge, these smells are not detected by any of the available CSS analysis tools.

Based on the syntax of CSS, we categorize CSS code smell types into four classes:

File-based smells: each CSS file is considered as a whole and is marked as a smelly file if more than a number of specific poor patterns occur in that file;

Rule-based smells: each CSS rule is considered as a code block and is marked as a smell if it is found to be smelly based on the smell detection criteria;

Selector-based smells: the selector part of a rule is considered as a whole and marked as a smell if any of its selector units are smelly;

Property-based smells: a declaration of a rule is marked as a smell if its property or its value is smelly;

3.1 Smells

Table 3.1 summarizes our proposed set of smells, their type, and detection criteria. In this subsection, we describe each of these smells.

[4] http://csswizardry.com/2012/11/code-smells-in-css/
[5] http://css-tricks.com/magic-numbers-in-css/
[6] http://www.blogherald.com/2006/09/08/css-tips-and-tricks/
[7] http://www.smashingmagazine.com/2007/05/10/70-expert-ideas-for-better-css-coding/

Table 3.1: Our proposed list of smells, detected by CSSNose and included in our empirical study.

CSS Smell	Detection Criteria	Type
None-External Rules	Embedded and inline CSS code	Rule-based
Too Long Rules	($\#Properties > \tau_p,\ \tau_p = 5$ (section 3.2))	Rule-based
Too Much Cascading	($\#SelectorUnits > \tau_s,\ \tau_s = 4$ (section 3.2))	Selector-based
High Specificity Values	($\#IDUnits > \tau_i,\ \tau_i = 1$ or $\#ClassUnits > \tau_c,\ \tau_c = 2$ or $\#ElementUnits > \tau_e,\ \tau_e = 3$) (section 3.2)	Selector-based
Selectors with Erroneous Adjoining Pattern	No white space between classes or IDs	Selector-based
Too General Selectors	Selectors equal to `html`, `head`, `title`, `body`, `div`, `header`, `aside`	Selector-based
Properties with Hard-Coded Values	Absolute property values using hard-coded/magic numbers	Property-based
Undoing Styles	Property values equal to 0 or 'none'	Property-based

3.1.1 Non-External Rules

CSS was designed as a means to achieve separation of concerns. CSS code itself should not be intertwined with HTML code. Thus, CSS rules should be defined in external files, and not in HTML documents. Defining CSS rules in external files separates presentation from HTML structure, which in turn enables a more efficient content authoring and sharing [7]. In addition, writing CSS in external style sheets enables code reuse, meaning that the same style sheet can be applied to different documents. This approach is ideal when the style is applied to many pages since the look and feel of the entire web application can be changed by modifying only one file. CSS code that is inline or embedded in HTML documents is a smell that adversely influences code reuse and maintainability.

3.1.2 Too Long Rules

Similar to too long methods in object-oriented languages [8, 9], CSS rules with too many declarations can result in complex styling code, which can become hard to maintain. In addition, when a rule applies too many styling properties to selected DOM elements, often other rules need to be written to undo some of the effects. Hence, we consider each CSS rule with the total number of properties greater than a specific threshold τ_p as a smell. The approach for determining the value of this threshold is fully described in section 3.2.

3.1.3 Too Much Cascading

Too much cascading in the selector part of a rule is a bad pattern since the DOM elements selected become too specific. We consider a selector as too specific with too much cascading when the total number of its selector units exceeds a specific threshold τ_s (see section 3.2). Figure 3.1 illustrates a too specific selector with too much cascading smell that consists of five selector units.

```
1 body div#test p.example {
2  color : blue;
3 }
```

Figure 3.1: Selector with Too Much Cascading

3.1.4 High Specificity Values

For selectors with too much cascading, the number of selector units determines whether the selector is smelly. However, a high specificity value can also indicate a CSS selector smell. Selectors with high specificity values are typically fragile and inhibit the reusability of the styling code with even minor changes in the structure of the DOM.

For instance, the selector `'#id1 #id2'` has only two selector units, but it has a specificity value (Equation 2.1) of 0200. Compare that to the selector `'body div table tbody tr td'`, which has six selector units, but a specificity value of 0006.

If *hasHighSpecificity(sel)* is a boolean function for identifying this type of too specific selectors with *sel* as its input selector and *abcd* as the specificity values of the selector units of *sel* (see Equation 2.1), then:

$$hasHighSpecificity(sel) = \begin{cases} true & \text{if } b > \tau_i \vee c > \tau_c \vee d > \tau_e \\ false & \text{otherwise} \end{cases}$$

where b, c, and d are the total number of ID, class and element units in sel, respectively, and τ_i, τ_c, and τ_e are thresholds for these values to be considered as a smell (see section 3.2).

Figure 3.2 shows selectors with high specificity values considering individual selector units and thresholds of $\tau_i = 1$, $\tau_c = 2$, and $\tau_e = 3$.

```
1 #id1 #id2 {
2   color : red;
3 }
4 .class1 .class2 .class3 {
5   border : 2px solid blue;
6 }
7 body div p span {
8   font-size : small;
9 }
```

Figure 3.2: Selectors with High Specificity Values

3.1.5 Selectors with Erroneous Adjoining Patterns

When the white space between two selector units is removed, the selector units are considered to be adjoined erroneously. Since these adjoined selectors are syntactically wrong, they are not recognized by the browser. Such selectors increase the CSS code size and browser workload without applying any styling properties to the DOM. Hence, this type of smell can be considered as an error type as well. Figure 3.3 depicts examples of selectors with erroneous adjoining patterns. As it can be seen, selector adjoining can happen between ID units, class units, ID/class units, and class/ID units.

3.1.6 Too General Selectors

This type of smell is the opposite of too specific selector smells. If the selector part of a rule selects all elements of the DOM based on their too general element type, the selector is considered too general. This type of selector is simply too broad and can ultimately lead to overridden style declarations

```
#id1#id2 {
  font-family: Arial;
}
.class1.class2 {
   color : gray;
}
#id1.class1 {
  line-height: 90%;
}
.class1#id1 {
  margin-left: 1cm;
}
```

Figure 3.3: Erroneous Adjoining Selectors

and undoing of CSS code. Figure 3.4 shows an example of this type of smell, which selects all `div` elements from the page.

```
div {
    background-color  : 00FF00;
}
body {
  color: silver;
}
```

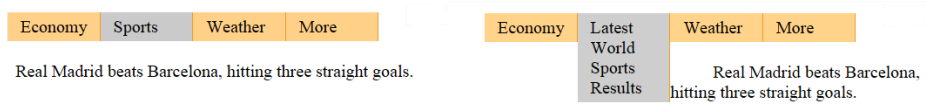
Figure 3.4: Too General Selector

We propose the following element types as too general when used as standalone selectors: `html`, `head`, `body`, `div`, `header`, `aside`; although this list is configurable in our smell detection engine.

3.1.7 Properties with Hard-Coded Values

This type of smell includes properties that have absolute or hard-coded numbers for their values. These numbers, also referred to as magic numbers, are not flexible meaning that when unexpected changes occur (e.g., in HTML structure or content), these absolute numbers fail to style the page properly and thus need to be adjusted. As a result, hard-coded values can cause unexpected problems in the visual layout of the application. For instance, Figure 3.6 shows CSS and HTML code snippets of the tabbed menu shown in Figure 3.5. In this example hard-coded values are defined for `padding`,

`border` and `width` properties. As a result, a longer text for Tab 2 results in an inappropriate page layout due to a fixed value (2 cm) for the width of each tab.



(a) Tabbed menu with short text in all tabs (b) Tabbed menu with longer text in Tab 2

Figure 3.5: Incorrect menu styling as a result of hard-coded CSS properties.

Instead of hard-coded values, it is a better practice to define property values relatively, e.g., using percentages.

3.1.8 Undoing Styles

Rules that define properties with values equal to 0 or 'none' reveal an undoing style pattern. This generally indicates that some of the CSS code has been applied too early or too broadly and now these rules are resetting the values to their default values. This undoing style pattern is considered a smell.

Figure 3.7 shows an example of this undoing style pattern. The border and padding properties are set for all `span` elements and their children through CSS inheritance. However, in line 6 these properties are reset. This style of coding requires extra effort to write and maintain.

3.2 Smell Thresholds

In our proposed set of smells, there are three types, namely Too Long Rules, Selectors with Too Much Cascading and Selectors with High Specificity Values, which have thresholds in their definition to be considered as a smell.

To determine the values for thresholds τ_p (Too Long Rules) and τ_s (Too Much Cascading), we conduct a small-scale study to take advantage of statistical evidence in addition to logical reasoning and development experience. To that end, we measure the number of CSS selector units and properties in 20 unique websites and compute their descriptive statistics. 10 of these websites are Alexa's[8] top 10 and the other 10 is selected randomly using a

[8] `http://www.alexa.com/topsites`

```
<style>
ul {list-style: none;}
ul li {float: left;}
ul li a {background-color: rgb(255,216,155);
color: black; display: block;
padding: 5px 15px; text-decoration: none;
border-right: 1px solid #f5ab36; width: 2cm;
}
</style>
<ul>
  <li><a href="url1">Economy</a></li>
  <li><a href="url2">Sports</a></li>
  <li><a href="url3">Weather</a></li>
  <li><a href="url4">More</a></li>
</ul>
```

Figure 3.6: Hard-coded values for the Menu shown in Figure 3.5

```
span {
  border: 5px solid black;
  padding : 5px 15px;
}
.contentpane {
  border: none;
  padding : 0 0;
}
```

Figure 3.7: Undoing Style CSS Code

random URL generator[9].

The results of this analysis are shown in Table 3.2. Using these results, we define:

$$\tau_p = \lceil Mean_p \rceil + 2 \quad , \quad \tau_s = \lceil Mean_s \rceil + 2 \tag{3.1}$$

where:

- $\lceil Mean_p \rceil$ is the smallest integer greater than or equal to $Mean_p$, where $Mean_p = \frac{\sum_{i=1}^{n} Mean_{p_i}}{n}$. n is total number of websites (i.e., 20) and

[9] http://www.randomwebsite.com/

Table 3.2: Descriptive statistics on the number of CSS selector units and properties, for 20 websites.

ID	Website	# properties in one rule				# selector units in one selector			
		Min	Mean	Median	Max	Min	Mean	Median	Max
1	Google	0	2.16	1	13	1	1.40	1	6
2	Facebook	0	2.42	2	15	1	1.96	2	7
3	Youtube	0	2.75	2	12	1	1.36	1	5
4	Yahoo	0	1.98	1	15	1	2.26	2	11
5	Baidu	0	2.16	1	15	1	1.75	1	5
6	Wikipedia	0	2.50	1	8	1	1.40	1	2
7	qq	0	3.97	2	14	1	2.14	2	8
8	Taobao	0	2.48	2	15	1	2.13	2	9
9	Live	0	1.93	2	9	0	1.58	1	6
10	Linkedin	0	2.38	2	16	1	2.72	3	9
11	Dvdmaniac	0	2.62	1	24	1	1.82	1	6
12	Whitestuff	0	2.06	2	10	1	1.77	2	4
13	Thespot	0	2.63	2	16	1	2.34	2	15
14	Javascriptkit	1	1.93	1	9	0	2.21	2	6
15	Angelfire	0	2.73	2	10	1	1.84	2	5
16	Megnut	0	2.44	2	11	1	2.17	2	7
17	Kickerclub	0	2.38	2	7	0	1.56	1	3
18	Bandwidthplace	0	1.81	1	16	1	2.07	2	9
19	Pensacolaswing	0	2.72	2	17	1	2.84	2	11
20	Onlamp	1	4.03	3	15	1	1.36	1	2
	Overall	0	2.5	2	24	0	1.9	2	15

$Mean_{p_i}$ is the average number of properties in each rule of the website with $ID = i(i = 1, ..., n.)$ which in turn is obtained by $Mean_{p_i} = \frac{\#\text{Total Properties in Website}(i)}{\#\text{Total Rules in Website}(i)}$

- τ_p is the smell threshold in terms of the number of properties in a CSS rule, which is computed as $\lceil 2.5 \rceil + 2 = 5$.

- $\lceil Mean_s \rceil$ is the smallest integer greater than or equal to $Mean_s$, where $Mean_s = \frac{\sum_{i=1}^{n} Mean_{s_i}}{n}$, n is the total number of websites (20) and $Mean_{s_i}$ is the average number of selector units in a selector for website with $ID = i$ *(i=1,...,n.)* which is computed as $Mean_{s_i} = \frac{\#\text{Total Selector Units in Website}(i)}{\#\text{Total Selectors in Website}(i)}$

- τ_s is the smell threshold in terms of the total number of selector units in one selector, which is determined as $\lceil 1.9 \rceil + 2 = 4$ in our study.

According to these thresholds, rules with total number of properties greater than five and selectors with more than four selector units should be reported as smells.

In order to define thresholds for the High Specificity Values smell, we take advantage of the definition of the specificity number (a, b, c, d) to compute the impact levels of different selector types. We ignore the left-most digit (a) since we have included non-external CSS rules in a separate category of smells. According to Equation 2.1, b, representing the total number of ID types, is the next most significant digit with the highest level of impact on the specificity value of a selector. c, representing the total number of class units has the second level of impact after b, and d, representing the total number of element units, has the lowest level of impact in the specificity value which should consequently have a larger threshold compared to number of ID and class units. Based on the above-mentioned rationality, we define thresholds as follows:

$$\tau_i = 1 \quad , \tau_c = 2 \quad , \tau_e = 3 \tag{3.2}$$

This means a selector has high specificity values if at least one of the following criteria is satisfied:

- Number of ID units in the selector part is greater than one;
- Number of class units in the selector part is greater than two;
- Number of element units in the selector part is greater than three;

3.3 Smell Detection

In order to detect the eight proposed smells, we have implemented a CSS smell detector called CSSNose. CSSNose is built as an extension to Cilla [1]. It uses dynamic crawling using Crawljax [10] to collect the CSS code of a given website. The collected CSS code is then processed using static analysis, to search for patterns of the proposed smell types, using the defined smell thresholds (See Table 3.1 and section 3.2). CSSNose then generates a report visualizing all instances of the detected CSS smells in the code.

3.4 Evaluation

To evaluate the accuracy of CSSNose, we compute its precision rate (PR) and recall rate (RR) as follows:

$$PR = \frac{TP}{TP + FP} \qquad and \qquad RR = \frac{TP}{TP + FN}$$

where TP, FP and FN are true positives, false positives and false negatives, respectively. PR identifies the rate of true smells among the detected smells and RR determines the effectiveness of the approach in identifying the true smells among all existing smells.

Computing these rates is a manual and time consuming task. To limit the amount of time needed for this step, we evaluate CSSNose on the 20 websites listed in Table 3.2.

We ran CSSNose on each of the 20 websites and examined the report generated for the eight proposed types of smells. We examined more than 350,000 lines of CSS code manually to count TP, FP, and FN. This process took us 11 days (10 working hours per day) in total.

Total number of true smells in the 20 websites under evaluation was 51,814. We did not find any false negatives, which was expected since CSSNose does not miss any lines of the CSS code in its static analysis process. We found 216 false positives in total (in 52,030 detected smells), which were mainly due to CSS parsing errors and could be resolved easily. For instance, a false positive detected by CSSNose is {`color: blu2e;`}. CSSNose mistakenly reports this property as a 'Properties with hard-coded values' smell. Table 3.3 fully depicts the precision rates of CSSNose in detecting our proposed set of smells for each of the 20 websites.

The overall recall and precision rates of CSSNose are 100% and 99.6%, respectively, which point to the high level of accuracy of the tool.

Table 3.3: Precision Rate (%): NA means the website does not have that type of smell

Website	Lines of CSS Code	Total # Detected Smells	Total # True Smells	Total # False Positives	Precision Rate (%) for Each Type of Smell								Overall Precision Rate (%)
					None-External Rules	Too Long Rules	Too Much Cascading	High Specificity Values	Erroneous Adjoining	Too General Selectors	Hard-Coded Values	Undoing Styles	
Google	36,552	962	954	8	100.0	100	100	100	100	100	98.2	100	99.2
Facebook	22,694	9,452	9,450	2	99.9	100	100	100	100	100	100.0	100	100.0
Youtube	25,976	2,677	2,665	12	100.0	100	100	100	100	100	98.6	100	99.6
Yahoo	39,583	10,011	9,919	92	97.9	100	100	100	100	100	98.9	100	99.1
Baidu	3,176	1,392	1,389	3	99.7	100	100	100	100	100	99.7	100	99.8
Wikipedia	113	31	30	1	100.0	100	NA	NA	NA	NA	91.7	100	96.8
qq	7,357	524	522	2	98.9	100	100	100	NA	100	99.6	100	99.6
Taobao	80,509	4,231	4,224	7	98.3	100	100	100	100	100	99.9	100	99.8
Live	2,484	722	714	8	100.0	100	100	100	100	100	98.6	100	98.9
Linkedin	45,851	4,705	4,681	24	NA	100	100	100	100	100	98.8	100	99.5
Dvdmanics	1,522	197	196	1	NA	100	NA	100	NA	100	99.1	100	99.5
Whitestuff	35,094	8,662	8,647	15	100.0	100	100	100	100	100	99.7	100	99.8
Thespot	3,172	612	608	4	100.0	100	100	100	100	100	98.6	100	99.3
Javascriptkit	1,621	429	420	9	100.0	100	100	100	100	100	96.0	100	97.9
Angelfire	3,012	952	947	5	NA	100	100	100	100	100	99.2	100	99.5
Megnut	609	103	96	7	100.0	100	NA	NA	NA	100	90.7	100	93.2
Kickerclub	11,186	2,224	2,222	2	100.0	100	100	100	100	100	99.9	100	99.9
Bandwidthplace	8,520	1,306	1,299	7	100.0	100	100	100	100	100	98.9	100	99.5
Pensacolaswing	4,800	156	156	0	NA	100	NA	NA	NA	100	100.0	100	100.0
Onlamp	19,430	2,682	2,675	7	100.0	100	100	100	100	100	99.5	100	99.7
Overall	353,261	52,030	51,814	216	99.7	100	100	100	100	100	98.3	100	99.6

Chapter 4

Empirical Study

In this chapter, we describe our empirical study design to answer the following research questions:

RQ1 How pervasive are CSS code smells in deployed web applications? What are the most prevalent CSS code smells?

RQ2 To what extent does each type of smell occur in CSS code of today's websites? What portion of CSS rules, selectors, and properties are smelly?

RQ3 Is there a correlation between CSS code characteristics and the number of smell types/instances?

4.1 Included CSS Smells

To address the research questions, we study 26 CSS smell types and validation violations in total. These include:

- The eight smells proposed in this work and detected by CSSNose (see Table 3.1 in chapter 3),
- 16 types of CSS smells detected by the open-source CSS analysis tool CSS Lint.
- Two CSS validation violations detected by W3C CSS Validator.

Table 4.1 presents a concise overview of these 18 additional smells. More detailed descriptions can be found in Appendix A and in the CSS Lint [2] and W3C CSS Validator [3] documentations online.

Table 4.1: List of smells and errors detected by CSS Lint and W3C CSS Validator included in our empirical study

Tool	CSS Smell	Type
CSS Lint	Empty Catch Rules: No property is defined ($\#Properties = 0$).	Rule-based
	Use of !important	Property-based
	Use of @import: Used to refer CSS files within other CSS files (nested CSS).	Rule-based
	IDs in Selectors	Selector-based
	Unqualified Attribute Selectors: They match all elements first and then check their attributes.	Selector-based
	Qualified Headings: Heading elements preceded by one or more other selector types.	Selector-based
	Already Defined Headings: Heading elements used more than once.	Selector-based
	Units for Zero Values: Values such as 0px or 0cm.	Property-based
	Box Model Size: Using height/width properties in conjunction with padding and border.	Property-based
	Not Using Shorthand Properties: Mazinanian et al [6] fully describes this type and duplicate properties.	Property-based
	Duplicate Properties	Property-based
	Unknown Properties: Property names with typos.	Property-based
	Negative text-indent: Negative value for text-indent property.	Property-based
	Underscore Hack: Use of (_) before the property name.	Property-based
	Too Many Web Fonts: Use of more than five `@font-face` in a file.	File-based

Tool	CSS Smell	Type
CSS Lint	Too Many font-size: Use of 10 or more `font-size` properties in a file.	File-based
W3C CSS Validator	Parse Error: Syntactical issues in lines of CSS code such as missing braces.	Rule-based
	Value Error: The value of a property is not recognized by the browser.	Property-based

4.2 Smell Detection Framework

To detect all the 26 smell types, we integrated CSSNose, CSS Lint, and W3C CSS Validator as a single smell detection framework, as depicted in Figure 4.1.

The framework expects a URL as input, which is used for crawling the website and collecting CSS data. The URL is also provided to W3C CSS Validator directly to check for CSS validation errors and warnings. CSSNose checks the extracted CSS code looking for smells discussed in chapter 3. The extracted CSS code is also saved into files and fed into CSS Lint. The final report contains instances of the 26 smell types and validation violations.

Our framework and empirical data are available for download at `http://salt.ece.ubc.ca/software/cssnose/`.

4.3 Experimental Objects

We conduct our empirical study on a list of 500 unique websites in total available in Appendix C. 350 websites are Alexa's[8] top sites and 150 are collected randomly using a random URL generator[9]. In total, we investigate 5,060 CSS files, consisting of more than 10 million lines of CSS code. Table 4.2 shows descriptive characteristics of our experimental objects in terms of the number of CSS files, Lines of CSS code, CSS rules, selectors, properties, and DOM states explored.

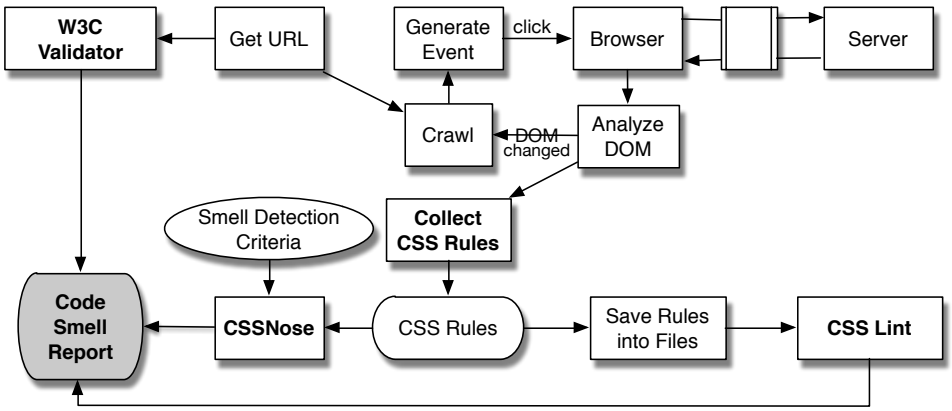


Figure 4.1: Overview of our overall smell detection framework

Table 4.2: Descriptive characteristics of the 500 experimental objects

	Min	Q1	Mean	Median	Q3	Max	Total
CSS Files	1	2	10	5	14	79	5,060
Lines of CSS Code	4	2,196	20,600	7,091	18,540	985,560	10,300,292
CSS Rules	1	169	1,372	658	1,635	16,732	686,220
CSS Selectors	0	201	1,800	808	2,191	20,378	900,049
CSS Properties	0	523	4,570	2,167	5,647	66,873	2,284,859
DOM States Crawled	1	1	25	20	50	50	12,385

4.4 Setup

The URL of each of the 500 websites is fed automatically into our smell detection framework (see Figure 4.1).

To collect the CSS code of each website, we configure Crawljax to click on the following DOM elements during crawling: `a`, `div`, `image`, `button`, `span`, and `input`. To constrain the time required, we set the maximum crawling depth to 3 similar to other studies [11]. We also set the upper limit for total number of DOM states to explore to 50, similar to other empirical studies [12]. We believe 50 pages is a large enough sample for extracting most (if not all) of the CSS code of a given website.

The code smell report generated for each website is saved and analyzed further using R[10]. Our empirical study was carried out during March–April 2014.

[10] `http://www.r-project.org`

4.5 Results

We discuss the results of the empirical study in response to our three research questions.

4.5.1 Pervasiveness (RQ1)

Our data shows that CSS smells are widespread in today's websites; 99.8% of the websites (i.e., 499 out 500) analyzed in our study contain at least one type of CSS smells.

Table 4.4 presents the pervasiveness of all the 26 smell types included in our study in terms of number of instances and percentage of sites exhibiting the smell type. As this table shows, CSS properties with *hard-coded values* and *undoing styles* are the most prevalent CSS code smells, occurring in 96% of the 500 websites, followed by *too long rules* and *IDs in selectors* in 94% and 90% of the websites, respectively.

4.5.2 Extent (RQ2)

We examine the concept of extent from different aspects to answer RQ2.

Number of Smell Instances/Types. Table 4.3 presents descriptive statistics of the total number of smell instances and types occurring in our 500 experimental objects.

According to this table, half of our experimental websites have at least 19 types of smells (out of 26 types of smells analyzed) in their CSS code. On average, there are 17.22 (out of 26) CSS smell types occurring in today's websites. In terms of the number of smell instances, half of the websites have at least 1,824 CSS smell instances and the average number of smell instances is 3,975 (0.33 smells per line of CSS code).

Table 4.3: Descriptive statistics of the total number of code smell types/instances (500 websites)

	Min	**Q1**	**Mean**	**Median**	**Q3**	**Max**
# Smell types	0	14.00	17.22	19.00	21.00	25.00
# Smells	0	481.00	3,975.16	1,824.50	4,859.00	65,990.00
# Smells per LOC	0	0.19	0.33	0.31	0.43	1.33

Ratio to Total. The pie chart in Figure 4.2 depicts the ratio of the total number of instances of each smell type to the total number of all detected smell instances. According to this chart, properties with *hard-coded values* contain 39.4% of the total detected smells, followed by *undoing styles*

Table 4.4: Pervasiveness of the 26 types of CSS smells in 500 websites

	Smelly Sites	
Smell Type	#	%
Non-External	371	74
Too Long	469	94
Too Much Cascading	359	72
High Specificity Values	410	82
Erroneous Adjoining Patterns	359	72
Too General	426	85
Hard-Coded Values	481	96
Undoing Styles	481	96
Empty Catch	414	83
!important	407	81
@import	67	13
IDs in Selectors	449	90
Unqualified Attribute Selectors	122	24
Qualified Headings	372	74
Already Defined Headings	361	72
Units for Zero Values	373	75
Box Model Size	443	89
Not Using Shorthand Properties	110	22
Duplicate Properties	305	61
Unknown Properties	148	30
Negative text-indent	255	51
Underscore Hack	243	49
Too Many Web Fonts	40	8
Too Many font-size	388	78
Parse Error	390	78
Value Error	368	74

(11.3%), *IDs in selectors* (8.6%), *non-external rules* (6.9%), and *parse errors* (5.1%).

Number of Smell Instances of Each Type. The second column of Table 4.5 (Total Number) presents descriptive statistics of the total number of detected code smells. According to this table, there are on average 1674 properties with *hard-coded values* and 420 properties with *none* or *zero* values in the 500 websites of our study.

Half of the 500 websites have at least 61 selectors with *high specificity*

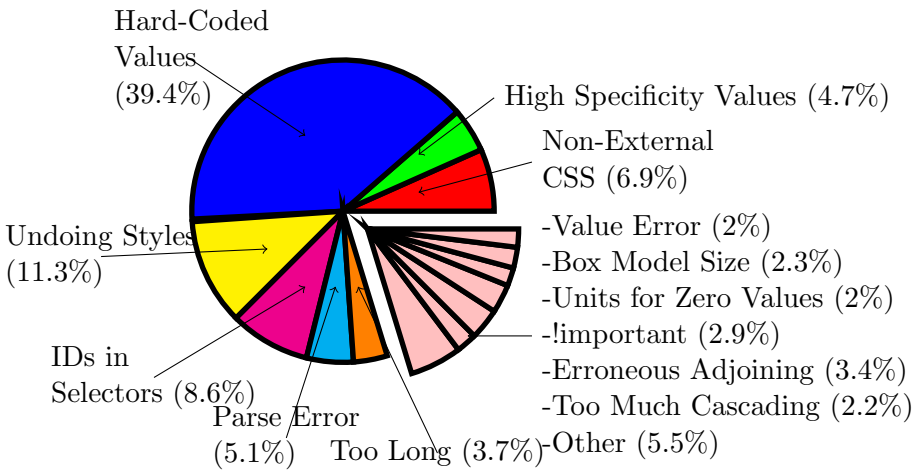


Figure 4.2: Numerical proportion of the detected smell types

values and there is a website with 15893 rules defined as *non-external*.

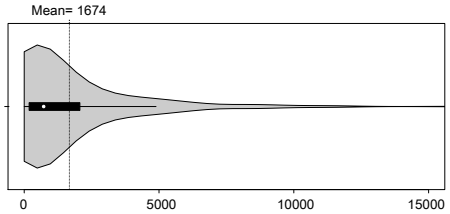
Figure 4.7 displays the distribution of smell instances for four smell types: *hard-coded values*, *undoing styles*, *IDs in selectors* and *non-external rules*. The violin plots for all other types of smells are shown in Appendix B.

Percentage of Smelly Rules/Selectors/Properties.

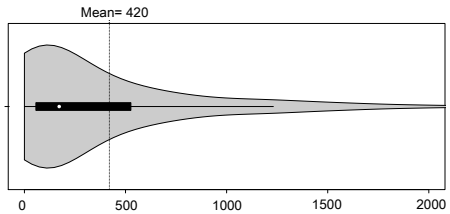
We further compute the percentage of the smelly code in our experimental objects with regard to each smell type individually to figure out the extent of smelly rules, selectors or properties. If st is a smell type and NS is total number of smells of type st, we then have:

$$\%SmellyCode(st) = \begin{cases} \frac{NS}{TotalRules} * 100 & \text{if } st \text{ is rule-based} \\ \frac{NS}{TotalSelectors} * 100 & \text{if } st \text{ is selector-based} \\ \frac{NS}{TotalProperties} * 100 & \text{if } st \text{ is property-based} \\ \frac{NS}{TotalFiles} * 100 & \text{if } st \text{ is file-based} \end{cases} \tag{4.1}$$

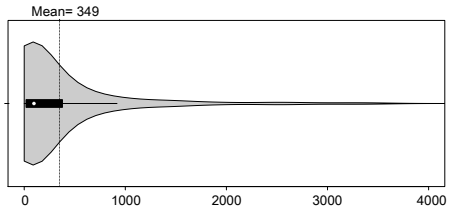
Using this function, on average 36% of property values are *hard-coded*. 23% of rules are *embedded* and 12% of CSS selectors have *high specificity values*. Also 35% of CSS files use *too many font-size* properties. The results are shown in the last column of Table 4.5 (Percentage).



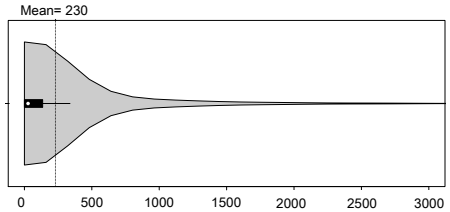
(a) Properties with hard-coded values



(b) Properties with undoing style values



(c) Selectors with IDs



(d) Non-external CSS rules

Figure 4.3: Distribution of the total number of smell instances of CSS smell types with the largest instance proportions among all detected smells.

Table 4.5: Descriptive statistics of detected code smells (500 websites)

	Total Number						Percentage					
Smell Type	Min	Q1	Mean	Med	Q3	Max	Min	Q1	Mean	Med	Q3	Max
Non-External	0	0	230	27	135	15,893	0	0	23	5	29	100
Too Long	0	15	132	62	180	958	0	7	12	11	15	65
Too Much Cascading	0	0	138	16	133	3,205	0	0	6	3	8	48
High Specificity Values	0	4	296	61	317	5,103	0	2	12	7	18	68
Erroneous Adjoining Patterns	0	0	218	23	171	5,934	0	0	9	4	11	100
Too General	0	2	12	7	13	241	0	1	2	1	2	43
Hard-Coded Values	0	182	1,674	722	2,065	47,826	0	29	36	36	43	100
Undoing Styles	0	57	420	172	527	3,638	0	8	12	10	12	100
Empty Catch	0	2	26	9	28	804	0	1	3	2	3	50
!important	0	2	123	22	109	3,080	0	1	4	2	3	100
@import	0	0	2	0	0	257	0	0	2	0	0	100
IDs in Selectors	0	18	349	97	378	8,770	0	5	22	16	34	100
Unqualified Attribute Selectors	0	0	6	0	0	877	0	0	1	0	0	5
Qualified Headings	0	0	54	11	57	839	0	1	3	2	4	36
Already Defined Headings	0	0	44	12	45	684	0	0	2	2	4	20
Units for Zero Values	0	0	63	9	46	1,693	0	0	2	0	2	31
Box Model Size	0	7	82	38	110	751	0	1	2	2	3	20
Not Using Shorthand Properties	0	0	1	0	0	95	0	0	1	0	0	18
Duplicate Properties	0	0	6	2	7	125	0	0	1	1	1	2
Unknown Properties	0	0	2	0	1	85	0	0	1	0	1	12
Negative text-indent	0	0	6	1	5	550	0	0	1	1	1	8
Underscore Hack	0	0	12	0	9	392	0	0	1	0	1	5
Too Many Web Fonts	0	0	0	0	0	3	0	0	2	0	0	100
Too Many font-size	0	1	3	1	3	37	0	6	35	25	50	100
Parse Error	0	1	62	16	64	1,439	0	0	3	0	0	100
Value Error	0	0	13	4	12	306	0	0	11	2	8	100

4.5.3 Correlations (RQ3)

To investigate correlations, we calculate the Spearman correlation coefficients [13] and p-values [14] between the total number of smell types/instances and different CSS code metrics, using R.

The Spearman correlation coefficient is a number between -1 and 1. 0 indicates no correlation between two data sets. 1 indicates a positive strong correlation and -1 shows a negative strong correlation.

CSS Metrics. The code metrics we investigate for correlations include number of CSS LOC, CSS files, CSS rules, selectors, properties, and class/element/ID units.

We also include three CSS code quality metrics defined by Keller and Nussbaumer [7] in our study. These metrics are defined as follows:

$$\text{Universality} = \frac{\#\ \text{ES}}{\#\ \text{Total Selectors}} \tag{4.2}$$

where `# ES` is the number of selectors ending with an element unit, e.g., `.news DIV`.

$$\text{Average Scope} = \frac{\sum_{i=1}^{m} \#\text{DOM elements selected by selector (i)}}{n * m} \tag{4.3}$$

where m is the total number of selectors and n is the total number of DOM elements.

$$\text{Abstractness Factor} = min(\text{Universality},\ \text{Average Scope}) \tag{4.4}$$

The output of these metrics are numbers between 0 and 1. The basic idea behind these CSS quality metrics is that the more general CSS selectors are (except for a limited number of too general selectors discussed in chapter 3), the higher the quality of the CSS code is in terms of reusability and maintenance.

Table 4.6 depicts descriptive statistics of universality, average scope, and abstractness factor in the 500 analyzed websites, computed automatically by our smell detection framework. The low average values of these metrics is another indication of the poor quality of CSS code in the wild.

Correlation Coefficients. Table 4.7 shows the Spearman correlation coefficients and p-values for the set of CSS code metrics compared with the smell

Table 4.6: Descriptive statistics of universality, average scope and abstractness factor (500 websites)

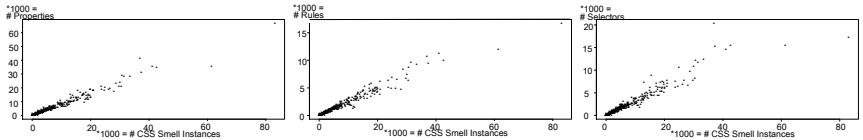
	Min	**Q1**	**Mean**	**Median**	**Q3**	**Max**
Universality	0	0.163	0.303	0.284	0.419	1
Average Scope	0	0.002	0.084	0.015	0.072	1
Abstractness Factor	0	0.001	0.058	0.013	0.064	1

instances and smell types. Figure 4.4 depicts their plots. Our results show that there is a strong positive correlation between total number of smell instances and number of CSS rules, selectors and properties ($corr = 0.96$, 0.92, 0.97, respectively). Total IDs, classes, and elements also show a positive correlation.

Table 4.7: Spearman correlation coefficients (corr) with p-values between CSS code smell instances/types and CSS code metrics

	# Smell Instances		**# Smell Types**	
Metric	**corr**	**p-value**	**corr**	**p-value**
Total Properties	0.97	$< 2.2e^{-}16$	0.51	$< 2.2e^{-}16$
Total Rules	0.96	$< 2.2e^{-}16$	0.52	$< 2.2e^{-}16$
Total Selectors	0.92	$< 2.2e^{-}16$	0.51	$< 2.2e^{-}16$
Total Class Units	0.90	$< 2.2e^{-}16$	0.50	$< 2.2e^{-}16$
Total Element Units	0.77	$< 2.2e^{-}16$	0.50	$< 2.2e^{-}16$
Total ID Units	0.68	$< 2.2e^{-}16$	0.38	$< 2.2e^{-}16$
Total CSS Files	0.45	$< 2.2e^{-}16$	0.36	$< 2.2e^{-}16$
Total Lines of CSS Code	0.40	$< 2.2e^{-}16$	0.13	$3.6e^{-}3$
Abstractness Factor	-0.23	$4.5e^{-}7$	-0.38	$< 2.2e^{-}16$
Average Scope	-0.16	$2.8e^{-}4$	-0.29	$3.3e^{-}11$
Universality	-0.15	$8.4e^{-}4$	-0.09	$3.9e^{-}2$

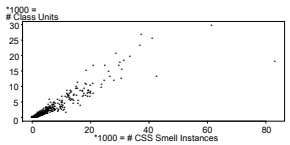
For the three quality metrics, namely universality, average scope, and abstractness factor, we do not see a significant correlation with smell instances in terms of magnitude, but the negative signs ($corr = -0.15$, $corr = -0.16$, $corr = -0.23$) indicate inverse correlations, which aligns with the concept behind these quality metrics, i.e., the higher the quality the less likely there will be smells in the code.



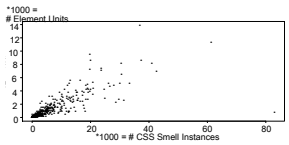
(a) Props: $corr = 0.97, p < 2.2e^{-}16$

(b) Rules: $corr = 0.96, p < 2.2e^{-}16$

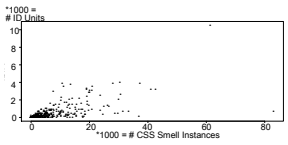
(c) Selectors: $corr = 0.92, p < 2.2e^{-}16$



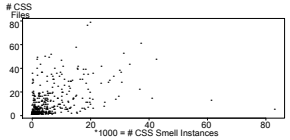
(d) Classes: $corr = 0.90, p < 2.2e^{-}16$



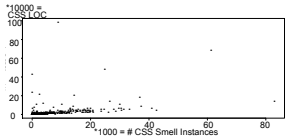
(e) Elements: $corr = 0.77, p < 2.2e^{-}16$



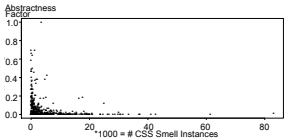
(f) IDs: $corr = 0.68, p < 2.2e^{-}16$



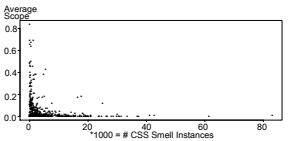
(g) CSS Files: $corr = 0.45, p < 2.2e^{-}16$



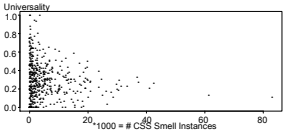
(h) CSS LOC: $corr = 0.40, p < 2.2e^{-}16$



(i) AbsFact.: $corr = -0.23, p = 4.5e^{-}7$



(j) Scope: $corr = -0.16, p = 2.8e^{-}4$



(k) Univ.: $corr = -0.15, p = 8.4e^{-}4$

Figure 4.4: Scatter plots of total number of CSS smell instances versus CSS metrics. *corr* represents the Spearman correlation coefficient and p is the p-value.

4.6 Unused Code Analysis

Unused CSS code increases file size and browser workload while having no effect on the web page design [1]. Hence, in addition to the 26 types of CSS smells that are detected by performing static analysis, we also include and analyze unused CSS code in this thesis. This type of smell is different from the previously mentioned smells in terms of the required type of analysis.

Unused CSS code which consists of unused selectors and properties can be placed in one of the following categories:

- Unmatched selectors which do not match any elements in the Document Object Model (DOM) and their properties;
- Matched but ineffective selectors and properties that match one or more elements of the DOM but are ineffective due to being overridden by style declaration of other selectors which have higher priority based on the specificity value or location;

Figure 4.5 designates some used and unused CSS code in an external CSS file, considering the DOM state in Figure 4.6.

```
1 div {
2 text-decoration: overline; /* matched but ineffective */
3  word-spacing : normal; /* used CSS */
4 }
5 p.today {
6 background-color: yellow; /* used CSS */
7 }
8 #weather {
9 color: red; /* matched but ineffective */
10             /* overridden by an inline style */
11 text-decoration: underline; /* used CSS */
12 }
13 p.update { /* unmatched selector */
14 font-family: "arial"; /* unused */
15 font-size: 100%; /* unused */
16 }
17 p {
18 background-color: gray; /* matched but ineffective */
19 }
```

Figure 4.5: External CSS File (example.css)

```
1 <head>
2 <link href="example.css" rel="stylesheet" type="text/css
    ">
3 </head>
4 <body>
5 <div id="weather" style="color:blue"> Weather Condition
6 <p class="today"> Current Weather </p>
7 </div>
8 </body>
```

Figure 4.6: DOM affected by CSS code in Figure 4.5

To detect unused code, we perform dynamic analysis and compare extracted and parsed CSS code on different crawled DOM elements. The core component of CSSNose is Cilla, a tool used in a previous study of unused CSS code [1]. Hence, our generated smell report includes statistics of unmatched and ineffective CSS. However, in order to get correct insight into CSS code of today's websites, we have separated results of unused code from other types of smells. The reason behind this decision is that CSSNose crawls web applications and automatically clicks on candidate clickables to collect CSS data. To detect unused code, it performs dynamic analysis to compare crawled DOM states with collected CSS. Since some of these DOM states are only accessible by entering human inputs from a real user such as usernames and passwords, some CSS code might be returned as unused incorrectly while it is actually used in a DOM state that CSSNose has not been able to access. This problem occurs due to the nature of dynamic analysis tools.

To mitigate this problem, we exclude those websites with required human inputs in analyzing results of our unused code smell to avoid false positives. For other types of smells, we have considered results of all 500 websites since we have used static analysis approach (no DOM access required) to detect all our smells except for the unused code. From our set of 500 websites, 313 websites need inputs from users. Hence, we limit analysis of our unused code study to the remaining 187 websites available in Appendix D. Characteristics of these 187 experimental objects are shown in Table 4.8.

4.6.1 Unused Code Pervasiveness

Unused CSS code is widespread in today's websites. Our data shows that 94% of the websites (i.e., 175 out of 187) analyzed in our unused code study

Table 4.8: Descriptive characteristics of the 187 experimental objects (Unused Code Study)

	Min	**Q1**	**Mean**	**Median**	**Q3**	**Max**
CSS Files	1	1	9	4	10	61
Lines of CSS Code	4	895	8,214	2,737	9,946	209,152
CSS Rules	1	64	707	203	726	10,009
CSS Selectors	0	71	943	263	971	15,491
CSS Properties	0	216	2,335	715	2,422	34,863
DOM States Crawled	1	1	23	16	50	50

contain at least one unused selector or property.

4.6.2 Unused Code Extent

Table 4.9 and Table 4.10 indicate the extent of unused selectors and properties in our 187 experimental websites. Our data confirms the results of a study of unused code performed by Cilla [1]. In this previous study, on average, 60% of selectors are reported as unused. In our experimental objects, on average 67.38% of selectors and 64.74% of properties are unused which shows an increase of almost 8% in the amount of unused selectors while considering a larger number of real world web applications. Table 4.11 also presents the percentage of size reduced in terms of bytes after applying the removal of unused selectors and properties.

Table 4.9: Descriptive statistics of unused CSS code in our 187 experimental objects (Total Number)

	Total Number					
	Min	Q1	Mean	Median	Q3	Max
Unmatched Selectors	0	35	762.36	168	812	13,647
Matched but Ineffective Selectors	0	0	8.39	3	10	126
Unused Selectors	0	40	770.75	172	818	13,674
Unused Properties	0	102	1,893.14	420	1,968	31,648

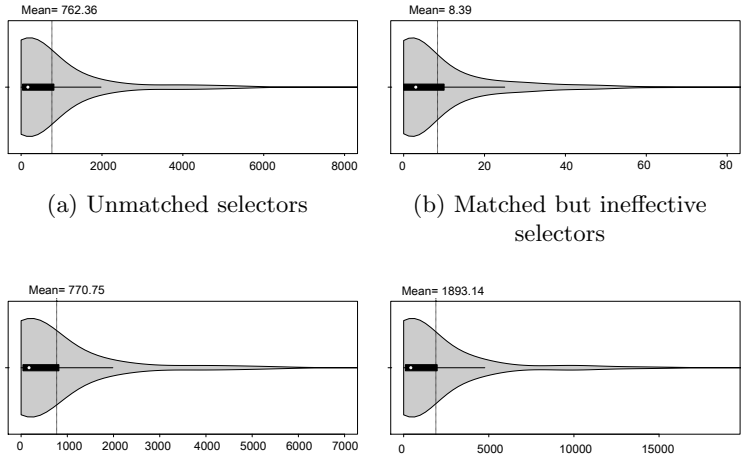
Table 4.10: Descriptive statistics of unused CSS code in our 187 experimental objects (Percentage)

	Percentage					
	Min	Q1	Mean	Median	Q3	Max
Unmatched Selectors	0	50.68	63.95	69.83	85.12	97.00
Matched but Ineffective Selectors	0	0	3.43	0.78	2.65	90.74
Unused Selectors	0	54.14	67.38	73.94	86.15	97.22
Unused Properties	0	46.65	64.74	70.73	87.51	97.38

Table 4.11: The size reduced in terms of bytes after applying the removal of unused code (187 websites)

	Min	**Q1**	**Mean**	**Median**	**Q3**	**Max**
Size Reduced (%)	0	31	41	45	53	88

Figure 4.7 shows violin plots of the distribution of the unused code based on the total number. As we can see, on average, 770.75 of selectors are unused in the 187 websites.



(a) Unmatched selectors

(b) Matched but ineffective selectors

(c) Unused selectors (unmatched + matched but ineffective selectors)

(d) Unused properties

Figure 4.7: Distribution of CSS Unused Code

4.6.3 Unused Code Correlations

Similar to our response to RQ3 for the 26 types of smells, we consider correlations between the extent of unused code and CSS metrics. We also include the number of crawled DOM states in our table of unused code correlations. The correlations between the total number of unused selectors/properties and CSS and DOM metrics are shown in Table 4.12.

Similar to other types of smells, unused code has strong correlations with the total number of rules, selectors and properties which is as expected since when there is more CSS code, it is more difficult for a developer to author and manage CSS code efficiently. The negative signs of the universality,

Table 4.12: Spearman correlation coefficients (corr) with p-values between unused CSS selectors/properties and CSS code metrics

	# Unused Selectors		**# Unused Properties**	
Metric	**corr**	**p-value**	**corr**	**p-value**
Total Rules	0.998	$< 2.2e^{-}16$	0.980	$< 2.2e^{-}16$
Total Properties	0.984	$< 2.2e^{-}16$	0.994	$< 2.2e^{-}16$
Total Selectors	0.996	$< 2.2e^{-}16$	0.979	$< 2.2e^{-}16$
Total Class Units	0.938	$< 2.2e^{-}16$	0.924	$< 2.2e^{-}16$
Total Element Units	0.860	$< 2.2e^{-}16$	0.854	$< 2.2e^{-}16$
Total ID Units	0.783	$< 2.2e^{-}16$	0.746	$< 2.2e^{-}16$
Total CSS Files	0.610	$< 2.2e^{-}16$	0.600	$< 2.2e^{-}16$
Total Lines of CSS Code	0.600	$< 2.2e^{-}16$	0.660	$< 2.2e^{-}16$
# DOM States Crawled	0.160	$1.6e^{-}2$	0.180	$1.6e^{-}2$
Abstractness Factor	-0.200	$6.6e^{-}3$	-0.190	$9.6e^{-}3$
Average Scope	-0.140	$6.9e^{-}2$	-0.120	$9.8e^{-}2$
Universality	-0.090	$2.3e^{-}1$	-0.100	$1.9e^{-}1$

average scope and abstractness factor coefficients also indicate an inverse correlation between these metrics and the amount of unused code.

Chapter 5

Smell Predictive Model

In this chapter, we propose a model for predicting the total number of CSS smells in a given website. We infer the predictive model from the large data set collected in our empirical study. This inferred model can be used as a metric to help developers gain insight in the quality of their CSS code.

Overall, in our approach we:

- consider the 500 experimental websites as a learning sample and use the collected empirical data as historical data (see chapter 4);
- apply a suitable fault prediction technique to our historical data to infer a predictive model (section 5.1);
- evaluate the accuracy of our predictive model on a test sample of 50 new websites (section 5.2);

5.1 Predictive Model Inference

In order to decide which predictive technique to apply on our historical data, we take advantage of the results of a comparative study on six predictive models conducted by Khoshgoftar and Seliya [15]. In their study, commonly used techniques are ranked in terms of their predictive accuracy. According to their results, the Least Absolute Deviation (LAD) method [16] is found to be the method with the lowest prediction error. Hence, we use LAD in this work.

Least Absolute Deviation (LAD). is a mathematical optimization technique for finding a function that closely approximates a set of data. To briefly formalize the problem, consider a set of data points (x_i, y_i), $i = 1, ..., n$. LAD attempts to find a function f such that $f(x_i) \approx y_i$ by minimizing the sum of the absolute values of the residuals (i.e., observed value − predicted value) that is $S = \sum_{i=1}^{n} |y_i - f(x_i)|$. In this formula, x_i, $i = 1, .., n$ is the set of explanators or predictor variables and y_i is the response variable which needs to be predicted. In case of multiple explanators, we need to

find $f(x_{ij}) \approx y_i$, for $i = 1, ..., n,\ j = 1, ..., m$ where n is total number of data points and m is the total number of predictor variables in the model.

Model Parameters. To decide which parameters should be included in our model, we consider CSS metrics that correlate with the total number of smells. Table 4.7 is a well-suited source of information for this purpose. Based on the empirically measured correlations, we select the following parameters for our model:

- Total Rules: the total number of CSS rules in the code of a website has a strong correlation with the total number of code smells ($corr = 0.96$).
- ID, class, and element units: these numbers are significantly correlated with the number of smells, as depicted in Figure 4.4. Hence, we include the ratio of IDs, classes, and elements with the respect to the total number of selectors in our model;
- Universality: we also include universality (see Equation 4.2) to take advantage of a CSS code quality metric in our model.[11].

Note that we could have used other metrics such as the average scope or abstractness factor in our model; the main reasons we choose universality is that this metric can be computed fast through static analysis and does not require the DOM elements affected in the scope of the CSS selectors (as required by the other metrics), which would need dynamic analysis to obtain.

Our CSP Model. To define our predictive model, we apply LAD on our data sets using the model parameters. After trying several function types on data points of the selected parameters, we infer a linear function for predicting the total number of CSS code smells per line of code, in a given website. Our CSP (CSS Smell Predictor) model is then defined as follows:

$$\text{CSP\_LOC} = \frac{\text{CSP}}{\text{LOC}} \tag{5.1}$$

where LOC is the total number of lines of CSS code and CSP is the total predicted CSS smells, which in turn is computed as follows:

$$\begin{aligned}\text{CSP} = 2.14(\#\text{Rules}) - 44.97(\text{Universality}) + 67.84(\frac{\#\text{ID units}}{\#\text{Selectors}}) \\ + 37.59(\frac{\#\text{Class units}}{\#\text{Selectors}}) - 24.25(\frac{\#\text{Element units}}{\#\text{Selectors}})\end{aligned} \tag{5.2}$$

[11] Using other metrics when proposing a new metric is a common practice. An example is the maintainability index [17]

By applying LAD, we obtained a positive coefficient for the total number of rules, which means that the more CSS rules we have, the more likely there will be code smells. A negative coefficient for universality Equation 4.2 is as expected and means that a higher universality value decreases the predicted total number of smells. The coefficient value for the ratio of class units (37.59) has a significantly smaller value compared to that of ID units (67.84) since using classes is a preferred pattern of writing CSS code.

Note that if the CSP is computed to be a negative number, our model returns 0. We have to mention that this is very rare in practice, since the positive factors in CSP are typically much larger than the negative factors; also in our evaluation of 50 websites, we did not witness any numbers below 0 (see Figure 5.1).

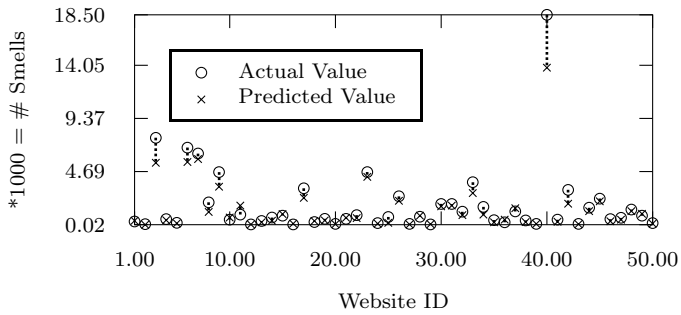
5.2 Model Evaluation

In order to evaluate the accuracy of our predictive model, we use a test sample consisting of 50 websites. These websites are collected randomly using a random URL generator [9] and are not in the list of 500 websites used in our empirical study.

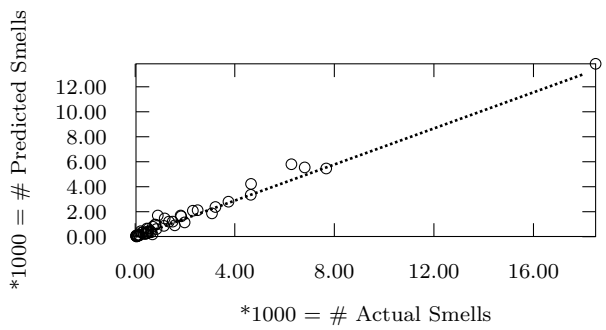
For each website, we measure the actual number of CSS smells as well as the predicted value computed by CSP. Figure 5.1 (a) depicts the actual and predicted values and their differences for the 50 websites in our test sample. As we can see, except for a few outliers, the difference between the actual and predicted values is small, which points to the accuracy of our CSP model. Figure 5.1 (b) shows actual and predicted values considering the diagonal line ($y = x$). If each website in our test sample is represented by a pair of coordinates (x, y) in which x is the actual number of smells an y is the predicted value, we have ($y = x$) for the websites with equal actual and predicted values, which is a perfect estimation. Hence, the closer the points are to the diagonal line, the smaller the differences between actual and estimated values, and consequently, the more reliable the model is. As we can see, for almost all of the websites in our test sample, the points are very close to the diagonal line, which is a good sign for the accuracy of our model.

Further, we use two well-known statistical techniques for evaluating predictive models, namely, residual plots and prediction errors.

Residual Plots. According to Garson [18], residuals analysis is a common approach in evaluating proposed models. A good predictive model has a random scatter plot of residuals (actual value − predicted value) against



(a) Total number of smells in a website: Actual values vs. values predicted by our model and their differences

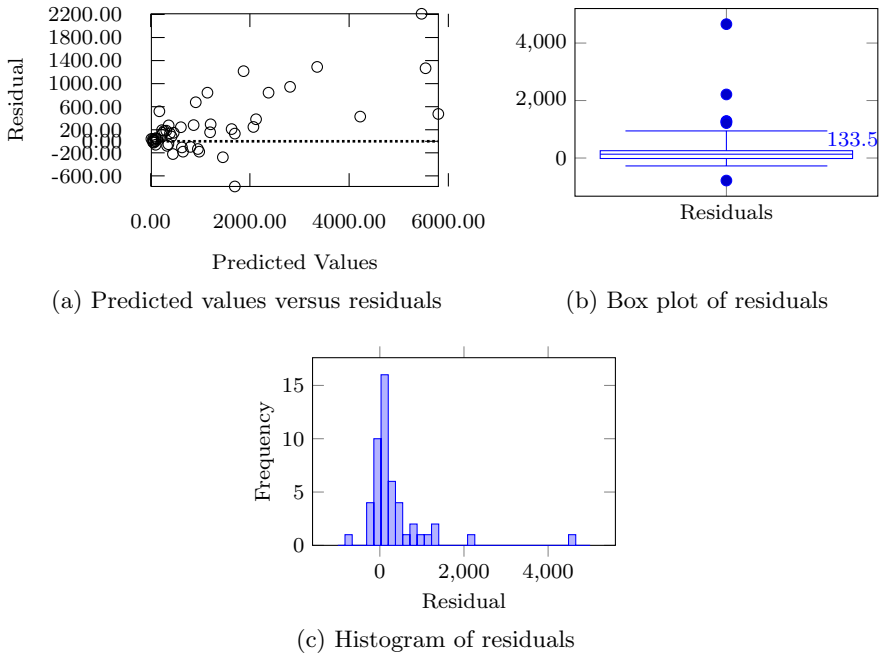


(b) Actual vs. predicted values

Figure 5.1: Actual versus predicated values

fitted (predicted) values, with no specific shape. Otherwise, there is some part of the data which is not covered and considered by the model. The median of the residuals should be zero or near zero and residual values need to be randomly negative and positive in order to form horizontal areas around the zero line. In addition, the histogram of residuals should depict a distribution close to normal.

Figure 5.2 depicts (a) a scatter plot, (b) a box plot, and (c) a histogram of the residuals of our proposed CSP model. In the scatter plot, we can see an almost random shape of residual points distributed as positive and negative values. This random shape is more obvious for the websites with less than 2000 number of smells. Since the residuals of our model do not follow a specific pattern and are randomly distributed, we can infer our model properly covers the population data. In addition, consistent distribution of positive and negative residual values, which is a sign of a good model, results in a zero or close to zero median of residuals. The median of residuals for our



(a) Predicted values versus residuals

(b) Box plot of residuals

(c) Histogram of residuals

Figure 5.2: Diagrams of residuals

model is 133.5 which is considered as a value close to zero considering the high number of CSS smells occurring in today's websites (see Table 4.3 and Table 4.5). The box plot shown in Figure 5.2 (b) is a better representation of the median and its difference with the zero value. Finally, a normal distribution of residuals is a result of real-valued random variables. Hence, the closer the distribution of residuals is to normal, the higher the accuracy of the predictive model is. The shape of the histogram of our residuals shown in Figure 5.2 (c) reveals a distribution close to normal.

Prediction Errors. There are several methods in statistics which compute errors in a predictive model. We use normalized root mean square error (NRMSE) [19], relative absolute error (RAE) [20], normalized mean absolute error (NMAE) [21], [22] and average relative error (ARE) [15] to evaluate our model. These errors are numbers from zero to infinity and are defined based on the differences between the actual and predicted values. The residuals $((\hat{y}_i - y_i))$ are numerators of the error fractions. Hence, the smaller the residuals are, the lower the error value is and the predictive model is more accurate.

Table 5.1: Prediction errors and their values for our model: $\hat{y}_i$, y_i and n are predicted values, actual values and the number of predictions, respectively. $\bar{y}$, y_{max} and y_{min} are the mean, maximum and minimum of the actual values.

Prediction Error	**Formula**	**Error Value of our Model**
NRMSE	$\frac{\sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}}}{y_{max} - y_{min}}$	0.05
RAE	$\frac{\sum_{i=1}^{n}\lvert\hat{y}_i - y_i\rvert}{\sum_{i=1}^{n}\lvert y_i - \bar{y}\rvert}$	0.24
NMAE	$\frac{\sum_{i=1}^{n}\lvert\hat{y}_i - y_i\rvert}{y_{max} - y_{min}}$	1.15
ARE	$\frac{\sum_{i=1}^{n}\lvert\frac{\hat{y}_i - y_i}{y_i + 1}\rvert}{n}$	0.33

The error formulas used and their values for our model are shown in Table 5.1. We can see that the error values of our model are small and close to zero, which indicates that differences between the actual number of CSS smells in a website and the number predicted by our model are not significant.

Overall, these evaluation results indicate that our proposed CSP model has a high accuracy in predicting the number of CSS smells.

Chapter 6

Discussion

In this chapter, we discuss some of the implications of our findings, limitations of our work, and threats to validity of our study.

6.1 Implications

Our study provides the largest empirical study on CSS to date in terms of the total number of websites analyzed and the diversity of CSS smell types. Our results reveal that poor CSS code is highly pervasive in today's websites. In terms of the most prevalent types of smells, properties with hard-coded values and undoing styles occur in 96% of the websites followed by too long rules and using IDs in selectors. Among these four most prevalent smells, three of them are proposed in this work and reported by our CSSNose smell detector tool; one of them (IDs in selectors) is detected by an existing tool CSS Lint. In addition, as shown in Figure 4.2, 72.4% of the total detected smells belong to the eight smell types proposed in our work. This points to the lack of proper tool support for developers in detecting CSS smells. CSSNose can help web developers to detect such prevalent smells during their development cycles.

The results of our study also indicate that properties with hard-coded values and undoing styles are not only among the most prevalent CSS code smells occurring in 96% of the websites, but they form the majority of the detected CSS code smells. These two smells form 39.4% and 11.3% of all detected smells, respectively. One of the reasons might be due to the fact that the total number of properties in CSS code of a website is usually larger than the total number of selectors, since each selector could have more than one defined property and hence, property-based smells comprise a higher proportion of CSS smells. However, we can see that there are other property-based smells such as using !important, underscore hack, and units for zero values which comprise a very small proportion of the detected smells. In addition, on average 36% of the properties in today's websites are hard-coded and 12% of the property values apply undoing style; that is a significant percentage of the smelly code. One explanation behind this

could be that developers are not aware of the inheritance characteristics of CSS code. Hence, they might focus on specific portions of the page and apply styling rules locally, necessitating the application of the style undoing pattern, later on.

Figure 6.1 depicts a snippet of actual CSS code taken from one of our experimental websites[12]. In the first line, hard-coded values are defined for the margin of all list elements with class attribute equal to 'entry'. The next rule adds a padding and undoes the margin values. Interestingly, this padding is also undone in the next rule (line 12).

```
1 body.entry li {
2   margin: 7px 0 8px 10
        px;
3 }
4
5 .enrtry ol {
6   padding: 0 0 0 35px;
7   margin: 0;
8 }
9
10 .entry ol li {
11   margin:0;
12   padding:0;
13 }
```

Figure 6.1: Poor CSS patterns in a real world CSS file

This is a simple example of poor CSS code in a real-world website. In this example all the CSS rules are defined in a single CSS file. The problem, however, becomes more severe when there are several CSS files with thousands of lines of code, making it more difficult for developers to write clean rules.

6.2 Code Smells

A list of code smells is never complete. In addition, some poor code patterns might be considered as acceptable code due to different needs of developers; This means that different detection criteria and thresholds can be defined for different scenarios. The set of eight smell types we propose in this work

[12] www.dvdmaniac.net

is the result of investigating a wide range of CSS development resources and code quality tools to include smell patterns that are (1) generally considered as a smell by the web development community, and (2) not already included in other CSS analysis tools. In terms of smell thresholds, we conducted a study on 20 websites to define unbiased thresholds based on average numbers, for too long rules and selectors with too much cascading. For selectors with high specificity values, we used the definition of the specificity number (Equation 2.1) to differentiate between ID units, class units, and element units, based on their level of impact on the specificity value of a selector. However, finding the most appropriate threshold is tricky and sometimes case-dependent. Despite the fact that we have used a combination of development experience, and statistical rationality to define reliable smell detection criteria, these thresholds can be modified in CSSNose, which in turn can influence the number of smells detected.

Sometimes some properties are considered as unused while they are equivalent to other properties and are intentionally added for the purpose of browser compatibility. For instance, in Figure 6.2, the same color has been defined as both hexadecimal and rgb values to deal with varying levels of browser support. Some browsers support rgb color while others do not. Hence, the following pattern is appropriate and common.

```
1 p {
2   color : #181818;
3   color : rgb(24,24,24);
4 }
```

Figure 6.2: Equivalent properties intentionally added for browser compatibility

We have considered the possibility of these equivalent properties in identifying duplicate properties detected by CSS Lint but not in our unused code detection. Hence, we might have some false positives in reporting unused properties. However, our detailed analysis of today's CSS shows that the number of this type of properties is not significant and can be easily ignored.

6.3 Our Predictive Model

To choose the predictive technique, we used the results of a previous comparison study [15] to select LAD. To infer the model parameters, we used our large empirical data. In addition, we used two well-known statistical

techniques, residual analysis and prediction errors, to evaluate the accuracy of our model. The results indicate a high level of accuracy for our CSP model. However, similar to all other predictive models, we cannot claim that our CSP model always produces accurate results that match the real number of CSS smells in a website. The goal of our predictive model is to easily provide developers with an estimation of the number of smells in their CSS code. CSS files can also be fed into this model and be prioritized based on the number of possible smells they include in their code. In case of limited resources, the static analysis can be performed on only those CSS modules which have a larger number of predicted CSS smells.

The discussion of our model also reveals a rationality behind the coefficients and defined relations.

6.4 Threats to Validity

Some of the threats to validity have been discussed above in this chapter. To make the results of our study as generalizable as possible, we have collected CSS data of 500 websites, with more than 10 million lines of CSS code. To obtain a representative sample, we have selected the URL of 350 Alexa's top websites in addition to 150 URLs selected randomly. We believe that studying 500 websites gives us a reliable dataset. One threat that is unavoidable in empirical studies that use online web applications is that the experimental objects and their CSS code might change in the future making exact replications of the study impossible. Nevertheless, we make the list of the analyzed 500 websites, the dataset, as well as our CSSNose and framework available for download.

Chapter 7

Related Work

We classify the related work into three categories.

7.1 Code Smells

There is a wide range of papers discussing code smells in general. Most of those have targeted the Java language. Murphy-Hill and Black [24] propose a smell detector tool for Java based on seven principles which are the basis of a highly effective smell detector. Hamid et al [25] conduct a comparative study on different smell detection tools. A set of refactoring guidelines for code smells in Java is proposed by Fowler et al [26]. Khomh et al [27] show that classes with code smells in object-oriented languages are more change-prone. Munro [31] helps developers to identify the characteristics of a bad smell in Java source code through the use of a set of software metrics and Simon et al [32] show that a special kind of metrics can support the decision of where to apply refactoring in the source code. Sjoberg et al [33] conduct a study to identify specific types of smells in Java that have a higher impact on code maintenance effort. Moha et al [29] describe and validate smell detection techniques for four types of design smells. Marinescu [30] presents a detection strategy by formulating metric-based rules that capture deviations from good object-oriented design principles.

In the web domain, Nguyen et al [28] discuss different types of embedded code smells in dynamic web applications, particularly focusing on server-side generated code and propose a tool to detect them. Milani Fard and Mesbah [23] propose a technique for detecting JavaScript code smells.

7.2 Smells in CSS Code

Keller and Nussbaumer [7] have defined a metric for CSS code quality. This metric is defined based on the level of code abstractness and is limited to the concept of selector scope. Mesbah and Mirshokraie [1] propose an automated analysis tool, called Cilla, to detect unused CSS code. Unused CSS code

pertains to code CSS rules that do not match any DOM elements as well as those that match but are ineffective due to rule overriding. CSSNose builds upon Cilla and extends it to detect CSS smells. Geneves et al [35] propose a technique based on tree logics which can be used in addition to existing runtime debuggers to ensure higher quality of CSS code. Mazinanian et al [6] define three types of duplication in CSS code and present a technique for discovering CSS refactoring opportunities to remove instances of those duplications.

Our work is the first large empirical study on CSS in terms of types of smells and number of experimental objects. Instead of considering only one specific type of smell, we have included 26 types of CSS smells in our study. In addition, our study is conducted on a set of 500 websites, which is the largest study so far.

7.3 Predictive Models

A wide range of traditional techniques, such as regression modelling [38], neural networks [39], and case-based reasoning [40], have been used in software engineering data analysis literature to derive usable and reliable models [41]. Many are based on empirical data. These models represent an approximation of the true functional form relating the independent and dependent variables on the basis of available input-output pairs [42]. Such models are capable of predicting a dependent variable, called the response variable, based on a set of independent variables used as predictors.

Afzal et al [44] define a predictive model of fault count for identifying software reliability growth based on genetic programming. Suresh et al [45] use object-oriented metrics to propose a fault predictive model based on regression and neural network. Having a similar goal, Emam et al [46] use object-oriented design metrics to predict faulty classes. Malhotra and Jain [47] use one statistical and six machine learning methods to predict models based on object-oriented metrics. Binkley and Schach [48] use coupling dependency metric as an independent variable to predict the total number of run-time failures and different maintenance measures in object-oriented languages and compare it with other metrics. Olague et al [49] empirically validate a set of three proposed object-oriented metrics to indicate their level of effectiveness in predicting fault-proneness in software components.

Our work is the first to propose and validate a predictive model for CSS code quality based on a large set of empirical data.

Chapter 8

Conclusions and Future Work

In this thesis, we have conducted a large empirical study on CSS code of 500 websites and answered research questions in terms of the pervasiveness and extent of different types of CSS code smells and their possible correlations with different code metrics. Our results indicate that today's CSS significantly suffers from inappropriate patterns and is far from well-authored code. In addition, we have proposed the first CSS code quality model derived from a large learning sample. This model helps developers to get an estimation of the total number of code smells in their CSS code.

Our work makes the following main contributions:

- We propose a set of eight new CSS code smells, which, to the best of our knowledge, are not detected by any other CSS maintenance tools;
- We present an automated technique, called CSSNose, to detect and spot our proposed set of smells in the code and evaluate its effectiveness and accuracy;
- We conduct a large-scale empirical study on 500 websites, 5060 files in total which consist of more than 10 million lines of CSS code, considering our proposed set in addition to 18 types of errors and warnings detected by two well-known CSS validation tools, CSS Lint and W3C CSS Validator to find the most prevalent CSS inappropriate patterns and their extent in today's websites;
- We investigate correlations between CSS code metrics and the total number of CSS smell types and instances;
- We propose a predictive model based on the findings of our large empirical study which is capable of predicting the total number of CSS in any given website;

As part of future work, we intend to investigate refactoring opportunities for the types of smells proposed in this paper and apply those that are presentation-preserving. In addition, we would like to enrich the evaluation of our predictive model by enlarging the test sample or using other possible model evaluation techniques and apply required enhancements.

Bibliography

[1] A. Mesbah and S. Mirshokraie, "Automated analysis of CSS rules to support style maintenance," in *Proceedings of the ACM/IEEE International Conference on Software Engineering (ICSE)*, pp. 408–418, IEEE Computer Society, 2012.

[2] W3Techs, "Usage of CSS for websites," 2014. `http://w3techs.com/technologies/details/ce-css/all/all`.

[3] G. J. Badros, A. Borning, K. Marriot, and P. Stuckey, "Constraint cascading style sheets for the web," in *Proceedings of the 12th annual ACM symposium on User interface software and technology (UIST)*, pp. 73–82, ACM, 1999.

[4] V. Quint and I. Vatton, "Editing with style," in *Proceedings of the 2007 ACM symposium on Document engineering (DocEng)*, pp. 151–160, ACM, 2007.

[5] H. W. Lie, *Cascading Style Sheets*. PhD thesis, Faculty of Mathematics and Natural Sciences, University of Oslo, 2005.

[6] D. Mazinanian, N. Tsantalis, and A. Mesbah, "Discovering refactoring opportunities in cascading style sheets," in *Proceedings of the ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE)*, p. 11 pages, ACM, 2014.

[7] M. Keller and M. Nussbaumer, "CSS code quality: a metric for abstractness; or why humans beat machines in CSS coding," in *Proceedings of the 2010 Seventh International Conference on the Quality of Information and Communications Technology (QUATIC)*, pp. 116–121, IEEE Computer Society, 2010.

[8] P. Meananeatra, S. Rongviriyapanish, and T. Apiwattanapong, "Using software metrics to select refactoring for long method bad smell,"

in *Proceedings of the 8th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTICON)*, pp. 492–495, IEEE Computer Society, 2011.

[9] P. Meananeatra, "Identifying refactoring sequences for improving software maintainability," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 406–409, ACM, 2012.

[10] A. Mesbah, A. van Deursen, and S. Lenselink, "Crawling Ajax-based web applications through dynamic analysis of user interface state changes," *ACM Transactions on the Web (TWEB)*, vol. 6, no. 1, pp. 3:1–3:30, 2012.

[11] B. He, M. Patel, Z. Zhang, and K. Chang, "Accessing the deep web," in *Communications of the ACM - ACM at sixty: a look back in time*, pp. 94–101, ACM, 2007.

[12] Z. Behfarshad and A. Mesbah, "Hidden-web induced by client-side scripting: An empirical study," in *Proceedings of the International Conference on Web Engineering (ICWE)*, pp. 52–67, Springer, 2013.

[13] S. H. Kan, *Metrics and models in software quality engineering*. Addison-Wesley Longman Publishing Co., 1994.

[14] D. J. Rumsey, *Statistics For Dummies, 2nd Edition*. For Dummies, 2011.

[15] T. M. Khoshgoftaar and N. Seliya, "Fault prediction modeling for software quality estimation: Comparing commonly used techniques," *Empirical Software Engineering*, vol. 8, pp. 255–283, 2003.

[16] P. Bloomfield and W. L. Steiger, *Least Absolute Deviations: THEORY, APPLICATIONS And Algorithms (Progress in Probability)*. Birkhuser Boston; 1 edition, 1984.

[17] P. Oman and J. Hagemeister, "Metrics for assessing software system maintainability," in *Software Maintenance, 1992. Proceerdings., Conference on*, pp. 337–344, IEEE, 1992.

[18] G. D. Garson, *General Linear Models: Multivariate, Manova, & Mancova*. Statistical Associates Publishers; 2012 edition, 2012.

[19] Y. Dodge, *The Oxford Dictionary of Statistical Terms.* Oxford University Press; 1 edition, 2006.

[20] X. R. Li and Z. Zhao, "Relative error measures for evaluation of estimation algorithms," in *Information Fusion, 2005 8th International Conference on*, vol. 1, pp. 8 pp.–, IEEE, 2005.

[21] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting*, vol. 22, pp. 679–688, 2006.

[22] P. Juszczak, D. M. J. Tax, and R. P. W. Duin, "Feature scaling in support vector data description," 2002.

[23] A. Milani Fard and A. Mesbah, "JSNOSE: Detecting javascript code smells," in *Proceedings of the International Conference on Source Code Analysis and Manipulation (SCAM)*, pp. 116–125, IEEE Computer Society, 2013.

[24] E. Murphy-Hill and A. P. Black, "Seven habits of a highly effective smell detector," in *Proceedings of the 2008 international workshop on Recommendation systems for software engineering (RSSE)*, pp. 36–40, ACM, 2008.

[25] A. Hamid, M. Ilyas, M. Hummayun, and A. Nawaz, "A comparative study on code smell detection tools," *International Journal of Advanced Science and Technology*, vol. 60, pp. 25–32, 2013.

[26] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: Improving the Design of Existing Code.* Addison-Wesley Professional; 1 edition, 1999.

[27] F. Khomh, M. D. Penta, and Y. Gueheneuc, "An exploratory study of the impact of code smells on software change-proneness," in *Proceedings of the 16th Working Conference on Reverse Engineering (WCRE)*, pp. 75–84, IEEE, 2009.

[28] H. V. Nguyen, H. A. Nguyen, T. T. Nguyen, A. T. Nguyen, and T. N. Nguyen, "Detection of embedded code smells in dynamic web applications," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 282–285, ACM, 2012.

[29] N. Moha, Y. Gueheneuc, L. Duchien, and A. L. Meur, "Decor: A method for the specification and detection of code and design smells," *Software Engineering, IEEE Transactions on*, vol. 36, pp. 20–36, 2010.

[30] R. Marinescu, "Detection strategies: metrics-based rules for detecting design flaws," in *Proceedings of the 20th IEEE International Conference on Software Maintenance*, pp. 350–359, IEEE, 2004.

[31] M. J. Munro, "Product metrics for automatic identification of "Bad Smell" design problems in java source-code," in *Proceedings of the 11th IEEE International Symposium on Software Metrics*, pp. 15–15, IEEE, 2005.

[32] F. Simon, F. Steinbruckner, and C. Lewerentz, "Metrics based refactoring," in *Proceedings of the Fifth European Conference on Software Maintenance and Reengineering*, pp. 30–38, IEEE, 2001.

[33] D. I. Sjoberg, A. Yamashita, B. C. Anda, A. Mockus, and T. Dyba, "Quantifying the effect of code smells on maintenance effort," *Software Engineering, IEEE Transactions on*, vol. 39, pp. 1144–1156, 2013.

[34] P. M. Marden and E. V. Munson, "Today's style sheet standards: The great vision blinded," *Computer*, vol. 32, pp. 123–125, 1999.

[35] P. Geneves, N. Layaida, and V. Quint, "On the analysis of cascading style sheets," in *Proceedings of the 21st international conference on World Wide Web (WWW)*, pp. 809–818, ACM, 2012.

[36] L. Briand and J. Wst, "Empirical studies of quality models in object-oriented systems," in *Advances in Computers*, pp. 97–166, Academic Press, 2002.

[37] G. Box, J. Hunter, and W. Hunter, *Statistics for experimenters: Design, Innovation, and Discovery.* Wiley-Interscience; 2 edition, 2005.

[38] M. L. Berenson, *Intermediate statistical methods and applications: A computer package approach.* Prentice Hall College Div; First Edition edition, 1982.

[39] L. V. Fausett, *Fundamentals of neural networks: Architectures, algorithms and applications.* Pearson; 1 edition, 1993.

[40] J. Kolodner, *Case-based reasoning.* Morgan Kaufmann, 1993.

[41] A. R. Gray and S. G. MacDonell, "A comparison of techniques for developing predictive models of software metrics," *Information and software technology*, vol. 39, pp. 425–437, 1997.

[42] M. Shin and A. L. Goel, "Empirical data modeling in software engineering using radial basis functions," *Software Engineering, IEEE Transactions on*, vol. 26, no. 6, pp. 567–576, 2000.

[43] Y. Luo, "Statistical defect prediction models for software quality assurance," Master's thesis, The concordia institute for information systems engineering, July 2007.

[44] W. Afzal, R. Torkar, and R. Feldt, "Prediction of fault count data using genetic programming," in *Multitopic Conference, 2008. INMIC 2008. IEEE International*, pp. 349–356, IEEE, 2008.

[45] Y. Suresh, L. Kumar, and S. K. Rath, "Statistical and machine learning methods for software fault prediction using ck metric suite: A comparative analysis," *Hindawi Publishing Corporation ISRN Sofware Engineering*, vol. 2014, p. 15 pages, 2014.

[46] K. E. Emam, W. Melo, and J. C. Machado, "The prediction of faulty classes using object-oriented design metrics," *Journal of systems and software*, vol. 56, pp. 63–75, 2001.

[47] R. Malhotra and A. Jain, "Fault prediction using statistical and machine learning methods for improving software quality," *Journal of Information Processing Systems*, vol. 8, pp. 241–262, 2012.

[48] A. Binkley and S. Schach, "Validation of the coupling dependency metric as a predictor of run-time failures and maintenance measures," in *Proceedings of the 20th international conference on Software engineering (ICSE)*, pp. 452–455, IEEE, 1998.

[49] H. Olague, L. Etzkorn, S. Gholston, and S. Quattlebaum, "Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes," *Software Engineering, IEEE Transactions on*, vol. 33, pp. 402–419, 2007.

Appendix A

CSS Code Smells Detected by CSS Lint and W3C CSS Validator Included in Our Empirical Study

A.1 Smells Reported by CSS Lint

CSS Lint is an open source CSS code quality tool which flags patterns that might be considered as errors or warnings and cause problems for developers by performing a static analysis of CSS code. This tool lists some rules that are used in order to report CSS code which is against these standard rules.

In this section, we fully describe 16 types of these errors and warnings included in our empirical study to enrich our CSS code quality investigation in terms of smell diversity.

A.1.1 Empty Catch Rules

As suggested by the name, these rules have no defined properties for their selectors. They increase size of the CSS file without applying any style to the web page.

A.1.2 !important

As discussed in chapter 2, CSS language provides a feature for developers to be able to give a higher priority to a desired property regardless of the location or specificity value by adding *!important* to its value. Using this feature specially in a high number of properties reveals that developers had trouble writing robust and effective CSS code and had to take reactive actions.

A.1.3 @import

These rules, shown in Figure A.1, are used to refer to CSS files within other CSS files. They disallow the possibility of parallel CSS downloads because while parsing the code, browser stops at each `@import` rule to download the specified file and does not continue to the next styling rules until the download has finished. Hence, using `link` tags is a better option for including style sheets which does not prevent from parallel downloading and can be considered as a refactoring opportunity for this type of smell.

```
1 /* smelly code: */
2 @import url('ex11.css');
3
4 /* refactored code: */
5
6 <link href="ex11.css" rel="
7 stylesheet"  type='text/css'>
```

Figure A.1: Using @import and its refactoring

A.1.4 IDs in Selectors

IDs decrease reusability of the CSS code due to their uniqueness. IDs can be used only once in a page and using them in selectors limits styling to a single element and disallows using benefits given by CSS. An ID is 255 times more specific than one class which means 256 chained classes are needed to override one ID.[4] Using more than one ID in one selector part shows even a more severe problem and lack of CSS development knowledge.

A.1.5 Unqualified Attribute Selectors

These selectors match all elements first and then check their attributes. This leads to low performance and increases workload. Similar to universal selectors which are noted with * and select all elements of the document[13], unqualified attribute selectors are too general to be used in the process of writing good CSS code. The following code gives two examples of this type of smells. Since browsers start evaluating selectors from right-to-left, first all the elements which match the attribute and its value inside the brackets

[13] http://www.w3.org/TR/CSS2/selector.html

are selected. Then an extra effort is required to choose only those elements with the ancestor attribute which is a class in our examples.

```
.selected [type=text] {
color : red;
}
.newtab [target=_blank] {
background- color : blue;
}
```

Figure A.2: Unqualified Attribute Selectors

A.1.6 Qualified Headings

These are heading elements `h1` to `h6` which are preceded by one or more other selector types. These heading elements should be defined as not scoped to a particular area in order to increase visual consistency and performance and also make maintenance easier. The heading elements which are not preceded by any other selector type allow CSS code to be reused and make the appearance of headings consistent in the entire web site. In addition, headings are built-in objects in object-oriented CSS (OOCSS)[14] which serve as blocks and facilitate reusing some parts of the code. Hence, they should preferably be defined as the only type in the selector part.

```
/* smelly code: qualified heading: */
.selected h1 {
color : red;
}

/* refactored code: */
h1 {
color : red;
}
```

Figure A.3: Qualified Heading

[14] http://www.oocss.org

A.1.7 Already Defined Headings

These are heading elements `h1` to `h6` which have been used more than once in the CSS code. Similar to qualified headings, defining these elements as selectors with different style declaration parts disables reusing them as built-in blocks in other parts of the code. They might also cause overridden style declarations.

A.1.8 Units for Zero Values

Zero values are recognized by browser both with and without units. Hence, units such as 'px' for zero values are redundant and removing them helps in saving bytes.

A.1.9 Box Model Size

Using height/width properties in conjunction with padding and border might cause misunderstanding and contradiction between developer's intention and actual behavior of the CSS code. The following code helps in understanding this type of smell.

```
1 /* smelly code: box model size: */
2 .selected {
3 height: 200px;
4 border: 2px solid blue;
5 padding: 10px;
6 }
7
8 /* refactored code: */
9 .selected {
10 box-sizing: border-box;
11 height: 200px;
12 border: 2px solid blue;
13 padding: 10px;
14 }
```

Figure A.4: Box model size and its refactoring

As we can see, the value for the height property is equal to `200px` while the actual height of the element is **224px** considering both padding and border values. This might be confusing specially for a new developer since she might forget about adding border and padding values to the explicit

value of the height and considers `200px` as the final height value which might affect other elements in the document adversely. In order to prevent from this problem, adding `box-sizing` property might be helpful. In the refactored code of Figure A.4, the browser is enforced to accept `200px` as the actual value of the height property while keeping border and padding inside the `200px` area leaving **176px** for content.

A.1.10 Not Using Shorthand Properties

This type of smell ignores the feature provided by CSS language which is the ability to replace a set of properties with one single property to avoid redundant code. Properties which are not shorthand increase file size and decrease readability of the code. We can see an example in the following code:

```
1 /* smelly code: not using shorthand properties: */
2 .selected {
3  background-color: rgb(255,0,255);
4  background-image: url(images/example.gif);
5  background-repeat: no-repeat;
6  background-position: top left;
7 }
8
9 /* refactored code: */
10 .selected {
11  background: rgb(255,0,255) url(images/example.gif)
12              no-repeat top left;
13 }
```

Figure A.5: Not using shorthand properties and its refactoring

List of shorthand properties is available online.[15]

A.1.11 Duplicate Properties

This type of smells includes the same properties with the same values and the same properties with different values which are being separated by at least one other property in the style declaration block. Figure A.6 depicts this type of duplication reported by CSS Lint.

[15] https://developer.mozilla.org/en-US/docs/Web/CSS/Shorthand_properties

```
1 .firstexample {
2 color: red;
3 word-spacing: normal;
4 color: red;
5 }
6 .secondexample {
7 color: red;
8 font-size: small;
9 color: green;
10 }
11 .thirdexample {
12 color: red;
13 font-family: "Times New Roman";
14 color: #FF0000;
15 }
```

Figure A.6: Duplicate Properties

CSS duplication in Figure A.7 is ignored by CSS Lint and consequently in our study since it is so common to use different values for equal consecutive properties intentionally to deal with varying levels of browser support for CSS.

```
1 .selected {
2 color: red;
3 color: #FF0000;
4 }
```

Figure A.7: Correct CSS: Not considered as duplicate

A.1.12 Unknown Properties

Property names with typos are placed in this category.

A.1.13 Negative text-indent

Sometimes it is required to replace a text element with an image. For instance, in order to show a logo on top of a page, `h1` tag can have the properties shown in Figure A.8.

```
1 <style>
2 h1 {
3 background-image: url('logo.gif');
4 text-indent: -9999px;
5 }
6 </style>
7 <h1> This heading is hidden! </h1>
```

Figure A.8: Using Negative text-indent

Since `h1` is a text element, the developer needs to add some text inside `h1` tag in the HTML document in order to be able to access the `h1` element. Then the text needs to be replaced with the logo image, so that it will be visible to only screen readers and not to sighted users.

Negative text-intent is used for this purpose. A large negative number such as -999 or -9999 is set as the value of the text-indent property to hide the text and show the image instead. However, using a large number such as -9999, a giant 9999px offscreen box needs to be drawn by the browser which is a significant workload and decreases performance.[16] In order to avoid this workload, we can apply the refactoring in Figure A.9.[17]

```
1 <style>
2 h1 {
3 background-image: url('logo.gif');
4 text-indent: 100%;
5 white-space: nowrap;
6 overflow: hidden;
7 }
8 </style>
9 <h1> This heading is hidden! </h1>
```

Figure A.9: Correct CSS: Not using negative text-indent

By this refactoring, text inside the `h1` tag is indented beyond the width of its container without wrapping and the overflowing content is hidden. Hence, it has prevented from the workload of drawing a 9999px box by adding just two more lines of CSS code.

Negative text-indents might also cause too long horizontal scrollbars

[16] http://www.sitepoint.com/new-css-image-replacement-technique/
[17] http://www.zeldman.com/2012/03/01/replacing-the-9999px-hack-new-image-replacement/

in right-to-left language pages. In order to avoid this long scrollbar, the `direction: ltr;` property can be added to the rule as a refactoring to set the text direction left-to-right. CSS Lint and consequently our study considers only those negative text-indent properties as smells which have not applied this refactoring.

A.1.14 Underscore Hack

It is a technique to apply CSS to Internet Explorer prior to version 7 by adding an underscore (_) before the property name. These browsers are not popular these days.[18]. Hence, this technique is outdated and is also ignored by other popular and modern browsers such as Chrome and Firefox.

A.1.15 Too Many Web Fonts

Despite the fact that web fonts have become popular to enable cross browser compatibility by declaring multiple font types in addition to the standard system fonts[19] [20], there is a disadvantage in using them. The referred font files by `@font-face` can be quite large and some browsers might block rendering while downloading them. Hence, using more than five web fonts in one style sheet is considered as a smell in CSS Lint and in our study.

A.1.16 Too Many font-size

Using too many `font-size` properties is a sign of poor CSS code with low reusability. Defining a separate `font-size` value for each element is a time-consuming task which can be mitigated by defining some standard `font-size` classes as reusable blocks and adding them to the HTML document as shown in Figure A.10.

A style sheet with 10 or more `font-size` properties in returned as a smell by CSS Lint.

A.2 Smells Reported by W3C CSS Validator

W3C CSS Validator is another well-known CSS code quality tool which compares CSS code to the standard CSS grammar and reports errors and warnings. In our study, we take advantage of the syntax analysis of the

[18] https://www.modern.ie/en-us/ie6countdown

[19] http://www.websitemagazine.com/content/blogs/posts/archive/2013/09/12/10-popular-google-web-fonts.aspx

[20] http://www.typeandgrids.com/blog/the-ten-most-popular-web-fonts-of-2013

```
1 .small {
2  font-size: 8px;
3 }
4 .medium {
5  font-size: 11px;
6 }
7 .large {
8  font-size: 14px;
9 }
```

Figure A.10: Correct CSS: Defining classes for different `font-size` values

source code performed by W3C CSS Validator to consider two types of validation errors in web applications.

A.2.1 Parse Error

These parsing errors can be caused by a wide range of syntactical issues in lines of CSS code such as missing braces, space between the ID prefix '#' and ID name and space between class prefix '.' and class name in the selector part of a rule.

A.2.2 Value Error

This type of error happens when the value of a property is not recognized by the browser. This might be due to a typo in the value name, space between value and its unit or using a wrong number of values for a property.

Appendix B

Diagrams for Distribution of Smell Instances of Different Types

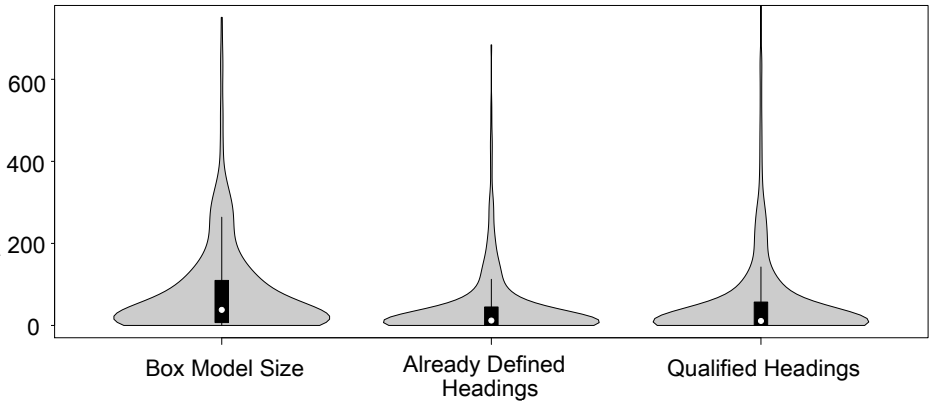


Figure B.1: Distribution of the total number of box model size, already defined headings and qualified headings smell instances in our 500 experimental objects

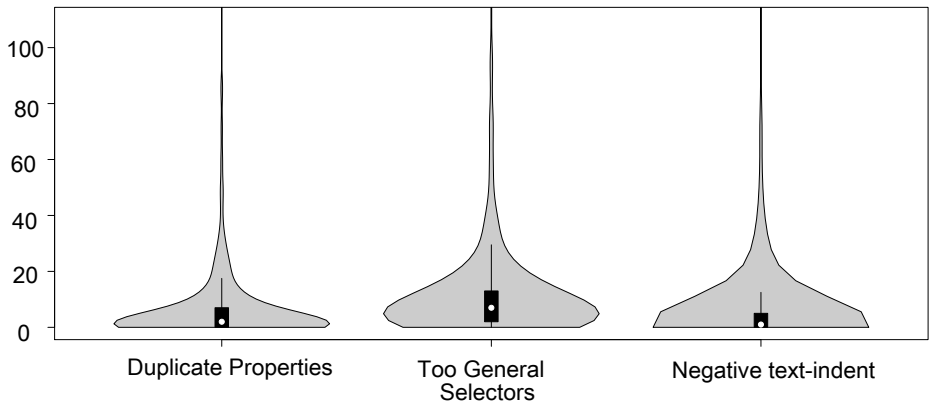


Figure B.2: Distribution of the total number of duplicate properties, too general selectors and negative text-indent smell instances in our 500 experimental objects

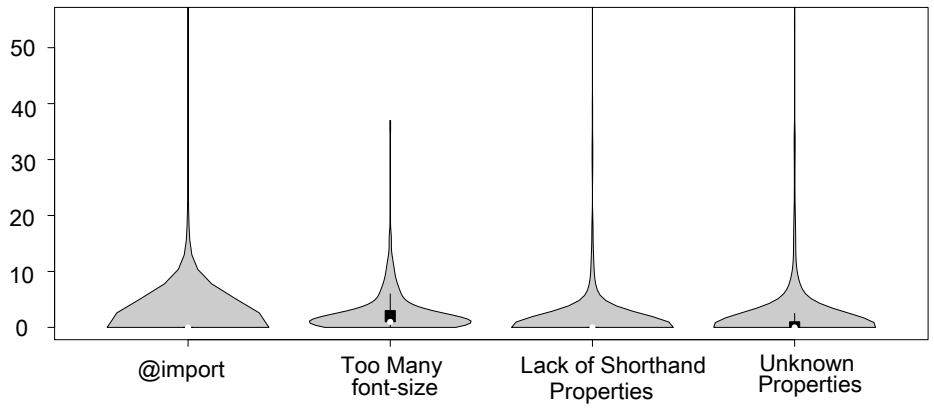


Figure B.3: Distribution of the total number of @import, too many font-size, lack of shorthand properties and unknown properties smell instances in our 500 experimental objects

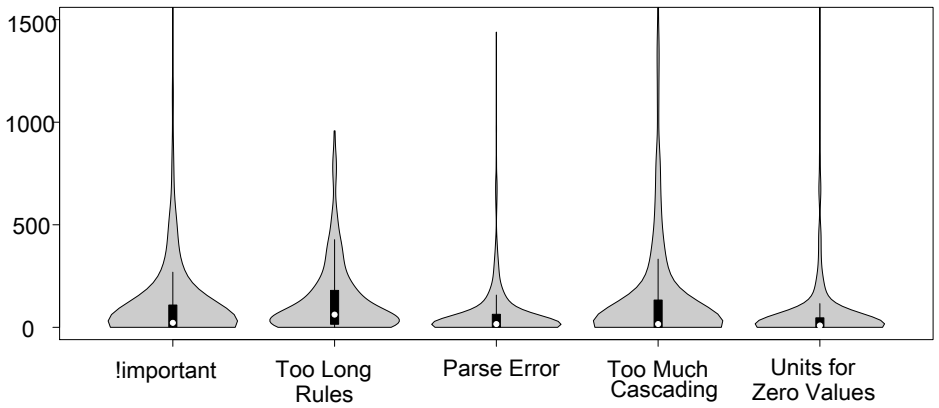


Figure B.4: Distribution of the total number of !important, too long rules, parse error, too much cascading and units for zero values smell instances in our 500 experimental objects

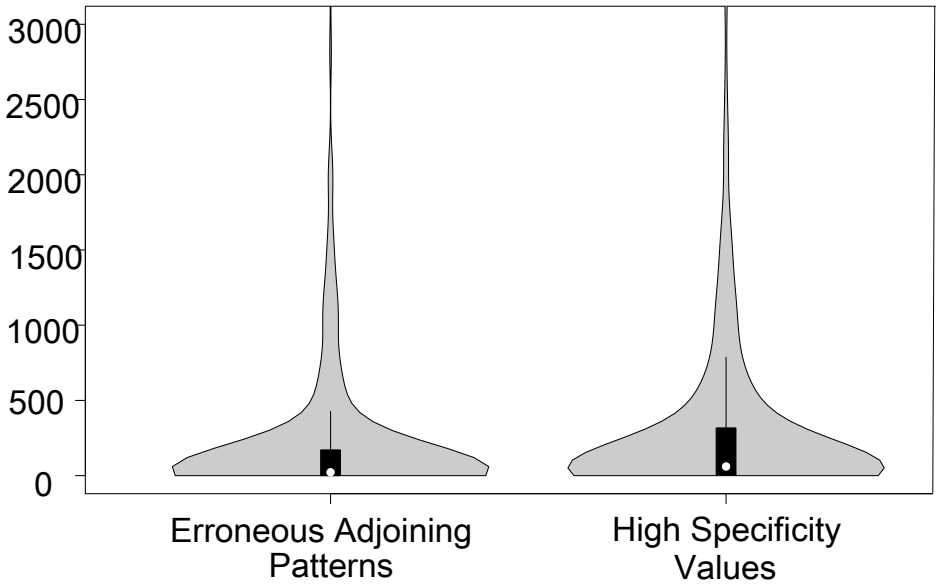


Figure B.5: Distribution of the total number of erroneous adjoining patterns and high specificity values smell instances in our 500 experimental objects

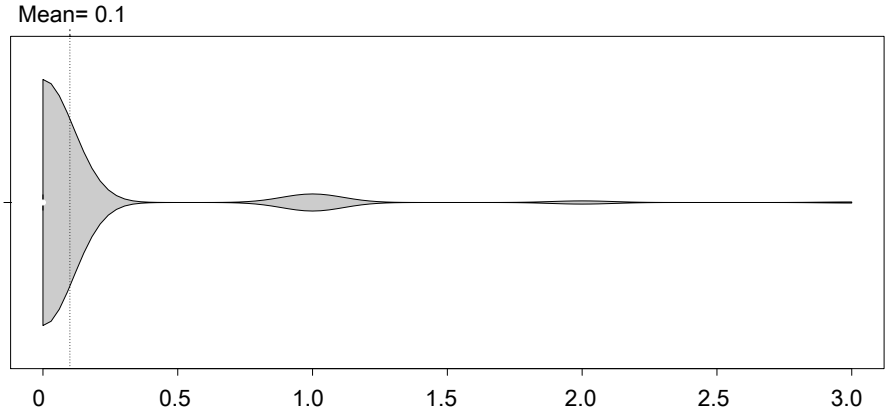


Figure B.6: Distribution of the total number of too many web fonts smell instances in our 500 experimental objects

Appendix C

List of 500 Websites in Our Empirical Study

C.1 Alexa's 350 Top Websites

1) http://www.google.com
2) http://www.facebook.com
3) http://www.youtube.com
4) http://www.yahoo.com
5) http://www.baidu.com
6) http://www.wikipedia.org
7) http://www.qq.com
8) http://www.taobao.com
9) http://www.live.com
10) http://www.linkedin.com
11) http://www.amazon.com
12) http://www.hao123.com
13) http://www.sberbank.ru
14) http://www.weibo.com
15) http://www.wordpress.com
16) http://www.yandex.ru
17) http://www.360.cn
18) http://www.bing.com
19) http://www.tmall.com
20) http://www.vk.com
21) http://www.ebay.com
22) http://www.sohu.com
23) http://www.pinterest.com
24) http://www.163.com
25) http://www.ask.com
26) http://www.soso.com
27) http://www.msn.com

28) http://www.tumblr.com
29) http://www.mail.ru
30) http://www.microsoft.com
31) http://www.xvideos.com
32) http://www.paypal.com
33) http://www.imdb.com
34) http://www.apple.com
35) http://www.xinhuanet.com
36) http://www.craigslist.org
37) http://www.imgur.com
38) http://www.xhamster.com
39) http://www.clixsense.com
40) http://www.123rf.com
41) http://www.bbc.co.uk
42) http://www.adcash.com
43) http://www.blogger.com
44) http://www.fc2.com
45) http://www.reddit.com
46) http://www.cnn.com
47) http://www.pcgames.com.cn
48) http://www.vube.com
49) http://www.go.com
50) http://www.adobe.com
51) http://www.odnoklassniki.ru
52) http://www.alipay.com
53) http://www.conduit.com
54) http://www.about.com
55) http://www.newgrounds.com
56) http://www.semrush.com
57) http://www.corriere.it
58) http://www.gmw.cn
59) http://www.flickr.com
60) http://www.aliexpress.com
61) http://www.godaddy.com
62) http://www.alibaba.com
63) http://www.pornhub.com
64) http://www.huffingtonpost.com
65) http://www.lady8844.com
66) http://www.netflix.com
67) http://www.ku6.com

68) http://www.thepiratebay.se
69) http://www.dailymotion.com
70) http://www.weather.com
71) http://www.vimeo.com
72) http://www.dailymail.co.uk
73) http://www.espn.go.com
74) http://www.foxsports.com
75) http://www.xnxx.com
76) http://www.rakuten.co.jp
77) http://www.indiatimes.com
78) http://www.themeforest.net
79) http://www.livejasmin.com
80) http://www.aol.com
81) http://www.uol.com.br
82) http://www.redtube.com
83) http://www.youporn.com
84) http://www.dropbox.com
85) http://www.nytimes.com
86) http://www.slideshare.net
87) http://www.globo.com
88) http://www.adf.ly
89) http://www.buzzfeed.com
90) http://www.china.com
91) http://www.directrev.com
92) http://www.mozilla.org
93) http://www.m2newmedia.com
94) http://www.ameblo.jp
95) http://www.fiverr.com
96) http://www.booking.com
97) http://www.livejournal.com
98) http://www.deviantart.com
99) http://www.yelp.com
100) http://www.sogou.com
101) http://www.flipkart.com
102) http://www.hootsuite.com
103) http://www.blogfa.com
104) http://www.oracle.com
105) http://www.etsy.com
106) http://www.outbrain.com
107) http://www.wikihow.com

108) http://www.avg.com
109) http://www.clkmon.com
110) http://www.stumbleupon.com
111) http://www.soundcloud.com
112) http://www.livedoor.com
113) http://www.w3schools.com
114) http://www.4shared.com
115) http://www.badoo.com
116) http://www.sourceforge.net
117) http://www.mediafire.com
118) http://www.torrentz.eu
119) http://www.liveinternet.ru
120) http://www.forbes.com
121) http://www.bankofamerica.com
122) http://www.addthis.com
123) http://www.aweber.com
124) http://www.foxnews.com
125) http://www.answers.com
126) http://www.indeed.com
127) http://www.bet365.com
128) http://www.gameforge.com
129) http://www.salesforce.com
130) http://www.hostgator.com
131) http://www.naver.com
132) http://www.espncricinfo.com
133) http://www.skype.com
134) http://www.github.com
135) http://www.softonic.com
136) http://www.statcounter.com
137) http://www.reference.com
138) http://www.spiegel.de
139) http://www.nicovideo.jp
140) http://www.shutterstock.com
141) http://www.allegro.pl
142) http://www.walmart.com
143) http://www.sharelive.net
144) http://www.mailchimp.com
145) http://www.tube8.com
146) http://www.gamer.com.tw
147) http://www.so.com

148) http://www.wix.com
149) http://www.zillow.com
150) http://www.wsj.com
151) http://www.popads.net
152) http://www.wellsfargo.com
153) http://www.goo.ne.jp
154) http://www.wordreference.com
155) http://www.photobucket.com
156) http://www.bild.de
157) http://www.pandora.com
158) http://www.warriorforum.com
159) http://www.pcpop.com
160) http://www.zedo.com
161) http://www.weebly.com
162) http://www.taringa.net
163) http://www.rutracker.org
164) http://www.php.net
165) http://www.ups.com
166) http://www.leboncoin.fr
167) http://www.mashable.com
168) http://www.goodreads.com
169) http://www.businessinsider.com
170) http://www.quikr.com
171) http://www.ucoz.ru
172) http://www.rediff.com
173) http://www.rambler.ru
174) http://www.gmx.net
175) http://www.telegraph.co.uk
176) http://www.domaintools.com
177) http://www.kaskus.co.id
178) http://www.comcast.net
179) http://www.ettoday.net
180) http://www.thefreedictionary.com
181) http://www.tianya.cn
182) http://www.wp.pl
183) http://www.avito.ru
184) http://www.ikea.com
185) http://www.goal.com
186) http://www.ndtv.com
187) http://www.uploaded.net

188) http://www.lpcloudsvr302.com
189) http://www.xcar.com.cn
190) http://www.baomihua.com
191) http://www.usps.com
192) http://www.pchome.net
193) http://www.moz.com
194) http://www.iqiyi.com
195) http://www.coccoc.com
196) http://www.thefreecamsecret.com
197) http://www.codecanyon.net
198) http://www.goodgamestudios.com
199) http://www.huanqiu.com
200) http://www.adrotator.se
201) http://www.twitch.tv
202) http://www.ci123.com
203) http://www.fedex.com
204) http://www.nbcnews.com
205) http://www.onclickads.net
206) http://www.it168.com
207) http://www.web.de
208) http://www.bitly.com
209) http://www.ehow.com
210) http://www.bitauto.com
211) http://www.delta-search.com
212) http://www.9gag.com
213) http://www.enet.com.cn
214) http://www.hp.com
215) http://www.daum.net
216) http://www.samsung.com
217) http://www.suning.com
218) http://www.myntra.com
219) http://www.varzesh3.com
220) http://www.sochi2014.com
221) http://www.scribd.com
222) http://www.olx.in
223) http://www.snapdeal.com
224) http://www.xuite.net
225) http://www.tmz.com
226) http://www.meetup.com
227) http://www.extratorrent.cc

228) http://www.java.com
229) http://www.doublepimp.com
230) http://www.4dsply.com
231) http://www.orange.fr
232) http://www.mercadolivre.com.br
233) http://www.infusionsoft.com
234) http://www.constantcontact.com
235) http://www.eazel.com
236) http://www.hulu.com
237) http://www.reuters.com
238) http://www.nih.gov
239) http://www.chinaz.com
240) http://www.xywy.com
241) http://www.rapidgator.net
242) http://www.detik.com
243) http://www.speedtest.net
244) http://www.libero.it
245) http://www.mobile01.com
246) http://www.clickbank.com
247) http://www.chaturbate.com
248) http://www.microsoftonline.com
249) http://www.bluehost.com
250) http://www.gsmarena.com
251) http://www.kooora.com
252) http://www.webmd.com
253) http://www.youjizz.com
254) http://www.histats.com
255) http://www.motherless.com
256) http://www.caijing.com.cn
257) http://www.in.com
258) http://www.xing.com
259) http://www.bestbuy.com
260) http://www.adnxs.com
261) http://www.americanexpress.com
262) http://www.cj.com
263) http://www.ad6media.fr
264) http://www.zippyshare.com
265) http://www.mywebsearch.com
266) http://www.nba.com
267) http://www.elpais.com

268) http://www.timeanddate.com
269) http://www.ign.com
270) http://www.hardsextube.com
271) http://www.techcrunch.com
272) http://www.zendesk.com
273) http://www.tinyurl.com
274) http://www.hdfcbank.com
275) http://www.snapdo.com
276) http://www.getresponse.com
277) http://www.lenta.ru
278) http://www.tagged.com
279) http://www.pof.com
280) http://www.force.com
281) http://www.cnzz.com
282) http://www.rt.com
283) http://www.irs.gov
284) http://www.douban.com
285) http://www.groupon.com
286) http://www.repubblica.it
287) http://www.siteadvisor.com
288) http://www.zimbio.com
289) http://www.seznam.cz
290) http://www.ero-advertising.com
291) http://www.w3.org
292) http://www.kakaku.com
293) http://www.elmundo.es
294) http://www.list-manage.com
295) http://www.xe.com
296) http://www.feedly.com
297) http://www.dell.com
298) http://www.ameba.jp
299) http://www.nydailynews.com
300) http://www.yaolan.com
301) http://www.lg.com.br
302) http://www.jrj.com.cn
303) http://www.upworthy.com
304) http://www.target.com
305) http://www.odesk.com
306) http://www.doorblog.jp
307) http://www.okcupid.com

308) http://www.ce.cn
309) http://www.zing.vn
310) http://www.habrahabr.ru
311) http://www.joomla.org
312) http://www.life.com.tw
313) http://www.surveymonkey.com
314) http://www.doubleclick.com
315) http://www.rbc.ru
316) http://www.naukri.com
317) http://www.jimdo.com
318) http://www.istockphoto.com
319) http://www.fbcdn.net
320) http://www.freelancer.com
321) http://www.blackhatworld.com
322) http://www.lenovo.com
323) http://www.flipora.com
324) http://www.webmoney.ru
325) http://www.justdial.com
326) http://www.gazeta.pl
327) http://www.latimes.com
328) http://www.jd.com
329) http://www.mama.cn
330) http://www.retailmenot.com
331) http://www.abcnews.go.com
332) http://www.4399.com
333) http://www.match.com
334) http://www.quora.com
335) http://www.informer.com
336) http://www.att.com
337) http://www.mysearchdial.com
338) http://www.sahibinden.com
339) http://www.ck101.com
340) http://www.ixxx.com
341) http://www.probux.com
342) http://www.capitalone.com
343) http://www.elance.com
344) http://www.icicibank.com
345) http://www.qtrax.com
346) http://www.issuu.com
347) http://www.twoo.com

348) http://www.ig.com.br
349) http://www.teensdigest.com
350) http://www.pch.com

C.2 150 Websites Collected Randomly

1) http://www.dvdmaniacs.net
2) http://www.whitestuff.com
3) http://www.thespot.nu
4) http://www.javascriptkit.com
5) http://www.angelfire.com
6) http://www.megnut.com
7) http://www.kickerclub.com
8) http://www.bandwidthplace.com
9) http://www.pensacolaswing.com
10) http://www.onlamp.com
11) http://www.subscene.com
12) http://www.jabong.com
13) http://www.empowernetwork.com
14) http://www.xgo.com.cn
15) http://www.irctc.co.in
16) http://www.jerseyjoeart.com
17) http://www.worldwatch.org
18) http://www.g4tv.com
19) http://www.filmthreat.com
20) http://www.jerwoodspace.co.uk
21) http://www.shirky.com
22) http://www.linklust.com
23) http://www.slgtm.com
24) http://www.freewarejava.com
25) http://www.accidentprone.com
26) http://www.onthebox.com
27) http://www.inmotionmagazine.com
28) http://www.dynamicdrive.com
29) http://www.indeterminate.de
30) http://www.oaklandmetro.org
31) http://www.explodingdog.com
32) http://www.gravityrec.com
33) http://www.interfacelift.com

34) http://www.plasticbag.org
35) http://www.parentfurther.com/technology-media
36) http://www.usdept-arttech.net
37) http://www.shehaal.com
38) http://www.moviemistakes.com
39) http://www.babylon.com
40) http://www.theyearofourlord.com
41) http://www.filmpicker.com
42) http://www.pterodactyl.info
43) http://www.lanacion.com.ar
44) http://www.writeslikeshetalks.com
45) http://www.xt-concept.com
46) http://www.coudal.com
47) http://www.maplestory.com
48) http://www.quiz1000.ca
49) http://www.blogjam.com
50) http://www.illruminations.com
51) http://www.bern-1914.org
52) http://www.earache.com
53) http://www.karlo.org
54) http://www.dublab.com
55) http://www.chapters.indigo.ca/home
56) http://www.satanslaundromat.com/sl
57) http://www.aspcr.com
58) http://www.regretless.com
59) http://www.thecutlerycollection.com
60) http://www.magicmarkerrecords.com
61) http://www.letourmentvert.com
62) http://www.paisleypop.com/store
63) http://www.roleplaygateway.com
64) http://www.crazyimages.com
65) http://www.richbarnard.com
66) http://www.48hourfilm.com
67) http://www.hxc.com
68) http://www.killingthebuddha.com
69) http://www.congressproject.org
70) http://www.michigan.gov
71) http://www.dailyscript.com
72) http://www.frank151.com
73) http://www.accesshiphop.com

74) http://www.thechaingang.com
75) http://www.goldenbonbon.com
76) http://www.prolific.org
77) http://www.chronicle.com/section/Home/5
78) http://www.flawedlogic.co.uk
79) http://www.zcommunications.org
80) http://www.hawthornestreetrecords.com
81) http://www.derstandard.at
82) http://www.jongevos.nl
83) http://www.mattmadden.com
84) http://www.saintcorporation.com
85) http://www.powerpage.org
86) http://www.ire.org
87) http://www.realstorygroup.com
88) http://www.browsehappy.com
89) http://www.notanalternative.com
90) http://www.hiropon-factory.com
91) http://www.overprint.com.ar
92) http://www.crailtap.com
93) http://www.filmus.is
94) http://www.boston.indymedia.org
95) http://www.hardcoremusic.com
96) http://www.petdance.com
97) http://www.eluniversal.com.mx
98) http://www.reprodepot.com
99) http://www.adbs.fr
100) http://www.blackhat.com
101) http://www.thefarrierguide.com
102) http://www.narnackrecords.com
103) http://www.readwrite.com
104) http://www.katespade.com
105) http://www.socialistvoice.org
106) http://www.davidrovics.com
107) http://www.verybusy.org
108) http://www.houseoftomorrow.com
109) http://www.the-business-blog.com
110) http://www.girlhacker.com
111) http://www.geektools.com
112) http://www.killrockstars.com
113) http://www.goateestyle.com

114) http://www.equalexchange.coop
115) http://www.dribbleglass.com
116) http://www.kuro5hin.org
117) http://www.aversionline.com
118) http://www.supah.com
119) http://www.42opus.com
120) http://www.katielinendoll.com
121) http://www.onfocus.com
122) http://www.wockerjabby.com
123) http://www.powerofcommunity.net
124) http://www.hugedomains.com
125) http://www.saigon.com
126) http://www.slipups.com
127) http://www.clubon.co.uk
128) http://www.funnygarbage.com
129) http://www.gordonandsmith.com
130) http://www.recircle.net
131) http://www.scul.org
132) http://www.badacidtrip.com
133) http://www.moddb.com
134) http://www.culturama.org
135) http://www.ac-et.com
136) http://www.burntsienna.nu
137) http://www.usounds.com
138) http://www.victorianhalloween.com
139) http://www.earthbrands.com
140) http://www.mreggs.com
141) http://www.fragdolls.com
142) http://www.commercialalert.org
143) http://www.consolidatedskateboard.com
144) http://www.nathanbeach.com
145) http://www.fortunecity.de
146) http://www.theslot.com
147) http://www.extrapepperoni.com
148) http://www.pointoflife.com
149) http://www.c3.hu
150) http://www.deliriouscool.org

Appendix D

List of 187 Websites in Unused Code Study

1) http://www.wikipedia.org
2) http://www.bing.com
3) http://www.soso.com
4) http://www.msn.com
5) http://www.microsoft.com
6) http://www.bbc.co.uk
7) http://www.go.com
8) http://www.about.com
9) http://www.thepiratebay.se
10) http://www.xnxx.com
11) http://www.china.com
12) http://www.directrev.com
13) http://www.mozilla.org
14) http://www.m2newmedia.com
15) http://www.ameblo.jp
16) http://www.sogou.com
17) http://www.w3schools.com
18) http://www.torrentz.eu
19) http://www.indeed.com
20) http://www.hostgator.com
21) http://www.sharelive.net
22) http://www.goo.ne.jp
23) http://www.wordreference.com
24) http://www.pandora.com
25) http://www.php.net
26) http://www.ups.com
27) http://www.ettoday.net
28) http://www.wp.pl
29) http://www.ikea.com
30) http://www.ndtv.com

31) http://www.lpcloudsvr302.com
32) http://www.moz.com
33) http://www.coccoc.com
34) http://www.thefreecamsecret.com
35) http://www.goodgamestudios.com
36) http://www.huanqiu.com
37) http://www.fedex.com
38) http://www.nbcnews.com
39) http://www.onclickads.net
40) http://www.delta-search.com
41) http://www.enet.com.cn
42) http://www.varzesh3.com
43) http://www.sochi2014.com
44) http://www.java.com
45) http://www.4dsply.com
46) http://www.nih.gov
47) http://www.gsmarena.com
48) http://www.bestbuy.com
49) http://www.mywebsearch.com
50) http://www.techcrunch.com
51) http://www.tinyurl.com
52) http://www.snapdo.com
53) http://www.lenta.ru
54) http://www.rt.com
55) http://www.irs.gov
56) http://www.repubblica.it
57) http://www.siteadvisor.com
58) http://www.w3.org
59) http://www.xe.com
60) http://www.feedly.com
61) http://www.upworthy.com
62) http://www.doorblog.jp
63) http://www.ce.cn
64) http://www.habrahabr.ru
65) http://www.joomla.org
66) http://www.doubleclick.com
67) http://www.rbc.ru
68) http://www.flipora.com
69) http://www.gazeta.pl
70) http://www.latimes.com

71) http://www.informer.com
72) http://www.mysearchdial.com
73) http://www.ixxx.com
74) http://www.qtrax.com
75) http://www.dvdmaniacs.net
76) http://www.javascriptkit.com
77) http://www.megnut.com
78) http://www.bandwidthplace.com
79) http://www.pensacolaswing.com
80) http://www.onlamp.com
81) http://www.jerseyjoeart.com
82) http://www.g4tv.com
83) http://www.filmthreat.com
84) http://www.jerwoodspace.co.uk
85) http://www.shirky.com
86) http://www.linklust.com
87) http://www.slgtm.com
88) http://www.freewarejava.com
89) http://www.accidentprone.com
90) http://www.onthebox.com
91) http://www.inmotionmagazine.com
92) http://www.dynamicdrive.com
93) http://www.indeterminate.de
94) http://www.oaklandmetro.org
95) http://www.explodingdog.com
96) http://www.gravityrec.com
97) http://www.interfacelift.com
98) http://www.plasticbag.org
99) http://www.parentfurther.com/technology-media
100) http://www.usdept-arttech.net
101) http://www.shehaal.com
102) http://www.theyearofourlord.com
103) http://www.pterodactyl.info
104) http://www.writeslikeshetalks.com
105) http://www.xt-concept.com
106) http://www.coudal.com
107) http://www.maplestory.com
108) http://www.quiz1000.ca
109) http://www.blogjam.com
110) http://www.illruminations.com

111) http://www.bern-1914.org
112) http://www.earache.com
113) http://www.karlo.org
114) http://www.dublab.com
115) http://www.satanslaundromat.com/sl
116) http://www.aspcr.com
117) http://www.regretless.com
118) http://www.magicmarkerrecords.com
119) http://www.letourmentvert.com
120) http://www.paisleypop.com/store
121) http://www.crazyimages.com
122) http://www.richbarnard.com
123) http://www.48hourfilm.com
124) http://www.hxc.com
125) http://www.killingthebuddha.com
126) http://www.congressproject.org
127) http://www.michigan.gov
128) http://www.dailyscript.com
129) http://www.frank151.com
130) http://www.accesshiphop.com
131) http://www.goldenbonbon.com
132) http://www.prolific.org
133) http://www.flawedlogic.co.uk
134) http://www.hawthornestreetrecords.com
135) http://www.derstandard.at
136) http://www.jongevos.nl
137) http://www.mattmadden.com
138) http://www.powerpage.org
139) http://www.realstorygroup.com
140) http://www.browsehappy.com
141) http://www.notanalternative.com
142) http://www.hiropon-factory.com
143) http://www.overprint.com.ar
144) http://www.crailtap.com
145) http://www.filmus.is
146) http://www.hardcoremusic.com
147) http://www.petdance.com
148) http://www.eluniversal.com.mx
149) http://www.reprodepot.com
150) http://www.blackhat.com

151) http://www.readwrite.com
152) http://www.socialistvoice.org
153) http://www.davidrovics.com
154) http://www.verybusy.org
155) http://www.houseoftomorrow.com
156) http://www.the-business-blog.com
157) http://www.girlhacker.com
158) http://www.geektools.com
159) http://www.killrockstars.com
160) http://www.goateestyle.com
161) http://www.dribbleglass.com
162) http://www.aversionline.com
163) http://www.42opus.com
164) http://www.katielinendoll.com
165) http://www.onfocus.com
166) http://www.wockerjabby.com
167) http://www.powerofcommunity.net
168) http://www.hugedomains.com
169) http://www.saigon.com
170) http://www.slipups.com
171) http://www.clubon.co.uk
172) http://www.funnygarbage.com
173) http://www.gordonandsmith.com
174) http://www.recircle.net
175) http://www.badacidtrip.com
176) http://www.usounds.com
177) http://www.victorianhalloween.com
178) http://www.fragdolls.com
179) http://www.commercialalert.org
180) http://www.consolidatedskateboard.com
181) http://www.nathanbeach.com
182) http://www.fortunecity.de
183) http://www.theslot.com
184) http://www.extrapepperoni.com
185) http://www.pointoflife.com
186) http://www.c3.hu
187) http://www.deliriouscool.org