CSS Corpus for Reproducible Analysis

Nico de Groot
Universiteit van Amsterdam
nico@nicasso.nl

Vadim Zaytsev
Raincode Labs
vadim@grammarware.net

Abstract

Reproducibility of research heavily depends on the availability of the datasets from the experiments — in the context of metaprogramming, the corpus of the code that was used to run the analyses and transformations. In the case of CSS, the problem is even more acute since the web is a constantly changing environment where the same address can refer to a frequently changing artefact.

In this report, we explain how we created a corpus of CSS files as a part of our project of building a framework for analysing style sheets. We also include two case studies of explanatory nature showing how style sheets from various websites go about coding conventions and about code duplication. We believe this work will be useful for other CSS researchers to compare techniques they develop, on a uniform yet realistic dataset.

1 Introduction

CSS, or Cascading Style Sheets, is the de facto standard in specifying the appearance of web pages. It is a language supported to some extent by all existing internet browsers and standardised by the World Wide Web Consortium — the leading authority in web technologies and standards [BÇHL11, eEG+11]. Even in the presence of other — better, modern, efficient, well-designed — alternatives, it remains the only industrially viable option for deployment of styles, leaving languages like SASS [CWE06] or LESS [SSP+09] to be used strictly on developers' side, if at all.

A typical style sheet in CSS could look like this:

```
1 h1, h2 {
2   font-style: italic;
3   font-size: 3em;
4   padding: 5px;
5 }
```

This sheet contains *one rule* with *two selectors* and *three declarations*. Each selector specifies one particular element to be matched with this style: in our simple example these are *element selectors*, other kinds of selectors including class selectors and ID selectors, as well as more complex pseudo-selectors for specifying the first child or the first line of the matched area. Each declaration assigns a value to a property, with the type of a value being determined by the particular property: a `padding`'s value is expected to be a length with a unit, but a `font-family` property expects a comma-separated list of names of individual fonts and font families.

CSS is an important element of the web development landscape, yet it is largely underrepresented in academic research. In a recent study we managed to cite all peer-reviewed papers ever published with CSS or "Cascading

Proceedings of the Seminar Series on Advanced Techniques and Tools for Software Evolution SATToSE2016 (sattose.org), Bergen, Norway, 11-13 July 2016, published at http://ceur-ws.org

Style Sheets" in the title [GZ16a]: 4 with general discussions, 2 case studies, 3 on pinpointing language shortcomings and improving on them, 5 on preprocessors, 2 on classification of syntactic errors, 7 on refactoring, 7 on analysis, 5 on security issues, 6 on IDE support. With the raising interest in spreadsheets and the maturity of them gaining acceptance among researchers, CSS is probably the most scarcely investigated industrially successful mainstream software language.

As part of a bigger project of building a framework to analyse CSS specifications [dG16b, dG16a], we have faced the challenge of empirical validation. We wanted to rely on a comprehensive corpus of CSS files, with reasonably high feature coverage numbers and potential for regression testing integration. This report collects many issues related to that particular part of our work, and exposes preliminary results. The work on the actual analysis tool and infrastructure is still ongoing and can be observed from the GitHub accounts of the authors.

Replications of website analysis papers are usually next to impossible since most modern active vendors change their applications continuously and deploying new versions up to 50 times a day [Sch14]. This means that providing the extensive list of websites used in the experiment, is not sustainable, since the actual CSS files behind those names would have changed hundreds and thousands times between the original experiment and its replication. One of the approaches is to provide a timed snapshot of a collection of web applications, and this is what this report focuses on.

As related work we can point the readers to Qualitas Corpus [TAD+10], a large collection of compilable software projects in Java; Atlantic Zoo [CTB+03], a versatile gathering of metamodels obtained by many means from mining papers to converting ontologies; or Grammar Zoo [Zay15], a grammarware-centred repository of artefacts of various nature containing knowledge about language structure. On an even more closely related note, both scale-wise and topic-wise, let us point to a recent paper by Mazinanian et al. [MTM14] on discovering refactoring opportunities in CSS. The dataset for the project was made publicly available [Maz14], which was greatly appreciated by replicators [PVZ16b]. However, as the effort by Punt et al. [PVZ16a] showed, the dataset had some issues related to crawling time glitches, crawling location specificity, file access misconfiguration, unavailability of cookies, and files being renamed. In our work we tried to combine the best we could learn from all these projects and approaches, which led to automated crawling of popular and acknowledged sources, with subsequent manual and tool-supported filtering, curation, standardisation. The result has over 0.5 MLOC of correct pretty-printed CSS, and can be used to test parsers and try out and compare techniques developed by CSS researchers.

The remainder of the report is organised as follows: section 2 explains the inclusion criteria, exposes details on how the corpus was composed and shows simple and advanced metrics calculated on it; section 3 sketches a case study of using the corpus with our work-in-progress framework to detect coding conventions; section 4 shows another case study about clone detection; section 5 draws preliminary conclusions and contemplates future endeavours.

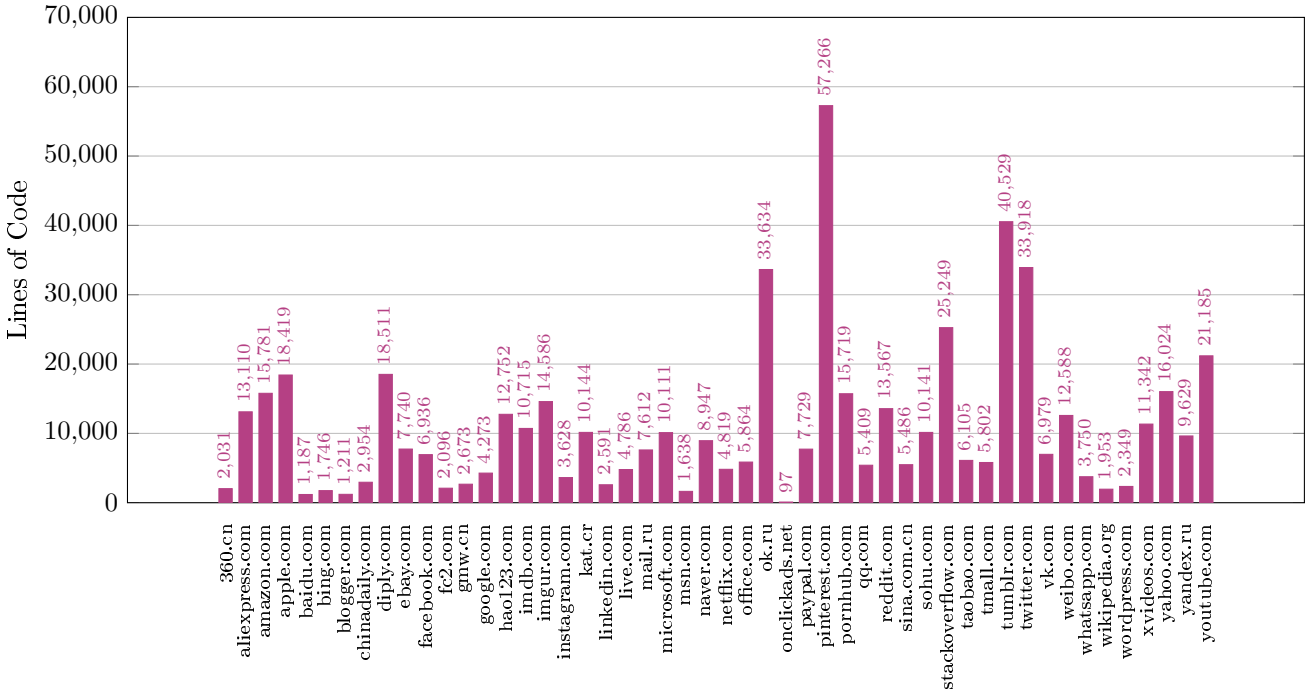
2 The Corpus: Contents, Selection, Metrics

The 50 style sheets for the corpus were picked from a selection of the most popular websites from the Alexa top 500 most popular sites on the web. Duplicate websites within the list such as `http://google.es` and `http://google.fr` have been ignored, since those would just result in the same style sheets multiple times. Furthermore, `http://t.co` has been ignored from the list since it is the link/redirect service of Twitter, and not a real website. Finally, `http://blogspot.com` is ignored as well since it refers to `http://blogger.com` which is already in the top 50. The final list covered the following websites:

360.cn	aliexpress.com	amazon.com	apple.com	baidu.com	bing.com
blogger.com	chinadaily.com	diply.com	ebay.com	facebook.com	fc2.com
gmw.cn	google.com	hao123.com	imdb.com	imgur.com	instagram.com
kat.cr	linkedin.com	live.com	mail.ru	microsoft.com	msn.com
naver.com	netflix.com	office.com	ok.ru	onclickads.net	paypal.com
pinterest.com	pornhub.com	qq.com	reddit.com	sina.com.cn	sohu.com
stackoverflow.com	taobao.com	tmall.com	tumblr.com	twitter.com	vk.com
weibo.com	whatsapp.com	wikipedia.org	wordpress.com	xvideos.com	yahoo.com
yandex.ru	youtube.com				

The actual style sheets from these websites have been downloaded using the CSS Stats tool [MJO14], which automatically extracts external style sheets as well as embedded CSS from web pages. The CSS provided by CSS

Figure 1: Amount of CSS used per website



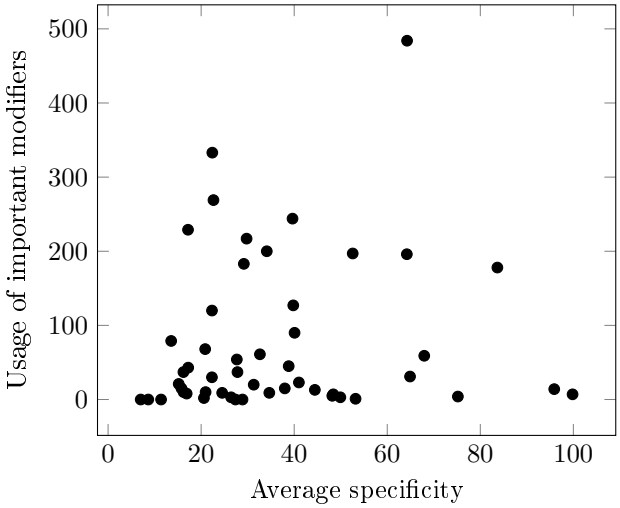
Stats does get pretty-printed, meaning that the CSS does not have its original lexical syntax any more. However, in the scope of our project this was beneficial, since most of the realistic CSS would have been minified otherwise. (Minification is a commonly adopted practice of code obfuscation aimed mainly at reducing the physical size of the code in bytes that need to be transferred to the client over the network; the secondary goal is usually to protect against reuse in derivatives). The pretty-printed CSS now allows for more accurate volume related analyses such as lines of code. Analysing strict coding conventions such as the existence of spaces in the right positions is not possible any more as the source code is not original, but this would have also been the case with minified deployed style sheets.

Volume metrics give an indication of the effort required to maintain the system. In general, the larger the project, the harder it will be to maintain. Figure 1 displays the 50 websites from the sample set with their amount of CSS (based on lines of code). The average amount of lines of code per website for the sample set is 10,866, with the median being 7,670. Some websites however are using much more, especially Pinterest, being almost six times the average with its 57,270 lines of code. The complete sample set has a total of 584,396 lines of code.

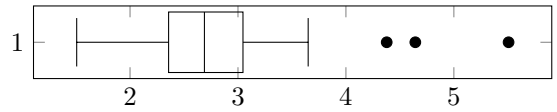
Applying **complexity** metrics [AMIO12] on the sample set resulted in more insights in the CSS. As can be seen in Figure 2b, the websites have on average about 2-3 declarations per rule set, 2.76 to be precise. Furthermore the average percentage of cohesive rule sets is 45,48%. The low average amount of declarations per rule set and the high amount of cohesive rule sets gives the impression that the CSS is developed with the "Single Responsibility Principle" in mind, allowing rule sets to be more easily reused since they only target specific properties.

Biçer et al. [BD16] introduce a total of **14 metrics** for CSS. Showing all 14 metrics would not provide any additional value, as most of these metrics are relatively similar. However, there is one metric which is not only interesting to implement, but also demonstrative for measuring the quality of the style sheets in the sample set. This metric is the Specificity metric, which measures the *specificity* of a selector. By presenting the specificity values for all selectors in a line chart, a specificity graph is created. In an ideal situation the specificity graph would have an upwards trend, where selectors get a higher specificity the later they are defined in the style sheet. So a style sheet should start with element selectors, as those have the lowest specificity, and work its way up to class selectors and then possibly have the style sheet end with some ID selectors, which have the highest specificity. The real goal of this graph is not to demonstrate that style sheets have all selectors perfectly in order based on their source order and specificity, as this would make for a extremely time consuming task. Small fluctuations in the specificity graph are not a big issue as long they do not stray too far away from the general

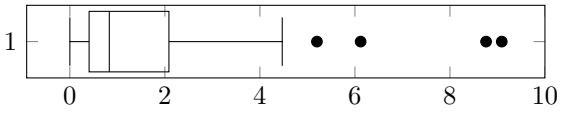
(a) Amount of important statements vs the average selector specificity



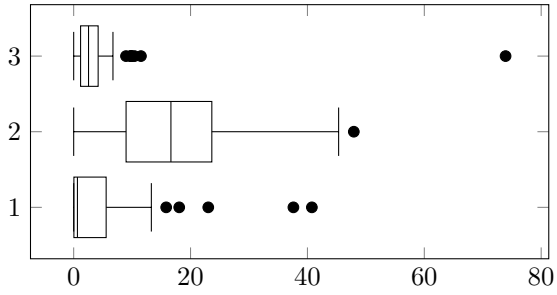
(b) Average amount of declarations per rule set



(c) Percentage of declarations with `!important` modifiers



(d) Percentage of cloned lines per type



upwards trend.

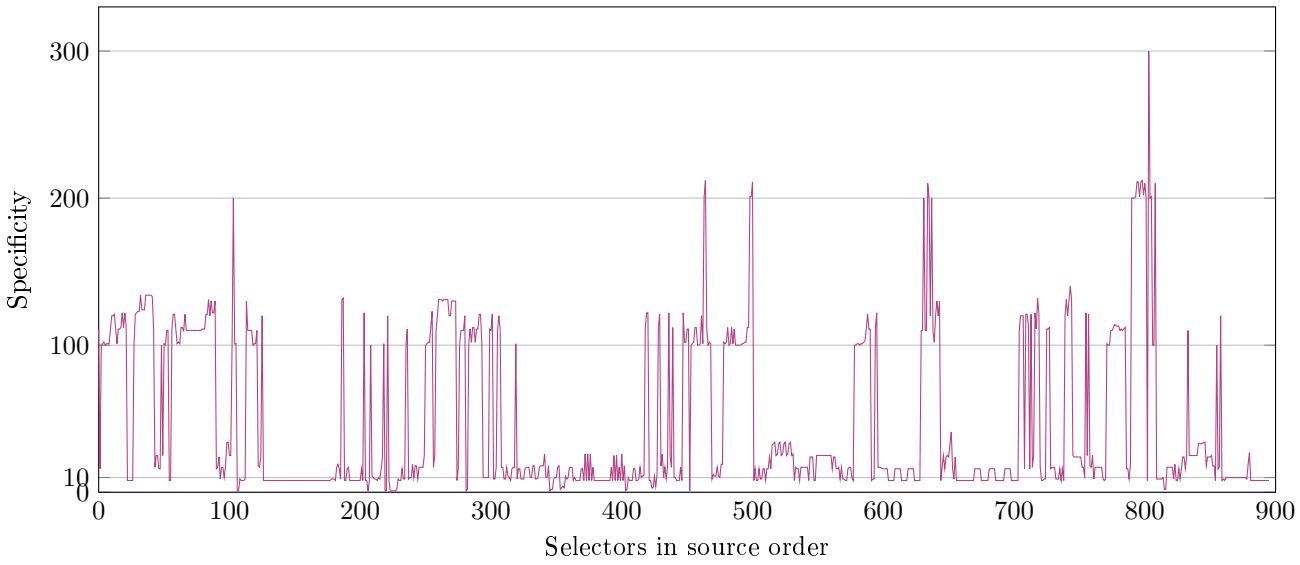
Having this upward trend in the specificity of selectors in the source file order does not impact the effect of the style sheet, since the actual order is only considered in case of conflicting selectors with equal specificity since then the source order will solve the conflict [eEG+11]. However, placing selectors in the style sheet in an upward trend, based on their specificity and source order, does make it easier to reason about the CSS. For example, if selectors with high specificity are placed at the beginning of the style sheet, and you later on have to change the presentation of those elements, you have either to overrule the specificity of the earlier defined selectors, or ensure they all have the same specificity.

An example of a specificity graph is shown in Figure 3, which is created using the style sheet of Whatsapp.com. Like the specificity graphs of most of the other websites from the sample set, the graph does not display a slowly increasing line. This can have multiple reasons, as for example all CSS from the websites of the sample set contain multiple style sheets which are all combined in one graph. This is due to the fact that during the downloading of the style sheets using the CSSstats tool, all style sheets have been merged into a single style sheet. Furthermore CSS preprocessors such as SASS or LESS could have been used to parse the CSS, placing rules in non-optimal positions. Our last hypothesis is that developers are not always completely familiar with the cascading characteristics of CSS.

What is interesting when looking at the specificity graph of Whatsapp.com, is that the style sheet immediately starts with very specific selectors. About the first 100 selectors seem to consist of mostly ID selectors and just some class and element selectors. This could affect the maintenance aspect of the style sheet, as possibly more specific selectors have to be used later on in the style sheet to overrule the already very specific base styles. This could also explain the high usage of the `!important` modifier for Whatsapp.com, as 9.09% of all declarations have applied it. The `!important` would allow the developers at Whatsapp.com for a quick and easy solution for solving cascading problems, even though it is considered a code smell [Zak11, GZ16a, Gha14].

The high amount of `!important` modifiers in Whatsapp.com, and its high average specificity, may give an impression that it could have positive correlation. This would be interesting due to the fact that a higher average specificity value would badly affect the maintainability of the style sheet, creating complex cascading related problems. Important modifiers would be a tempting "solution" for developers to use when the average specificity is high, as those are a quick and dirty way to solve these kinds of problems. However, this hypothesis has been refuted after a little probe, which results are shown in Figure 2a. An explanation for this outcome could be that websites which mostly use higher specificity selectors, will simply keep creating selectors with even higher specificity values, increasing the average specificity. As long as no low specificity selectors are used, no major cascading issues are likely to occur therefore not increasing the temptation for developer to use the `!important` modifier.

Figure 3: Specificity graph for Whatsapp.com



3 Case Study: Analysing Coding Conventions

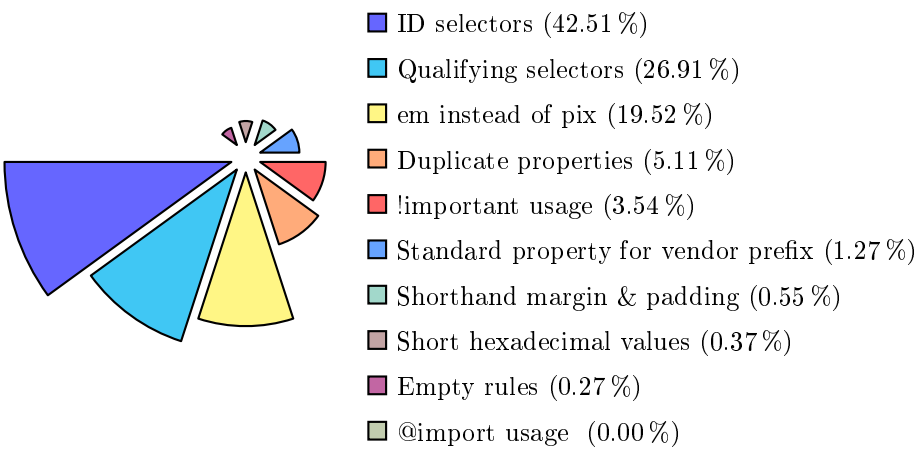
One of the analyses that are possible to implement within our framework is checking whether developers have applied coding conventions correctly. Checking if a semicolon is present after each declaration, if short hexadecimal values are used, or that a vendor-prefixed property is followed by a standard property [GZ16a], is all possible. Since there are also other tools that check coding conventions for CSS, we will compare our implementations for some coding conventions to theirs, and analyse how much more efficient our model is in conducting such analysis. Finally, a selection of the following ten coding conventions will be validated on the sample set, providing additional insights in the quality of the CSS [GZ16b]:

- Use short hexadecimal values (Performance)
- Use the shorthand margin and padding property (Performance)
- Disallow empty rules (Possible error)
- Do not use id selectors (Maintainability)
- Require standard property with vendor prefix (Compatibility)
- When possible, use em instead of pix (Accessibility)
- Disallow duplicate properties (Possible error)
- Disallow @import (Performance)
- Avoid using !important (Maintainability)
- Avoid qualifying ID and class names with type selectors (Performance)

The conventions were taken from open-source communities, companies and CSS professionals. They regard possible errors, compatibility, accessibility, maintainability, and performance [GZ16b]. Coding conventions related to lexical details such as required locations of spaces, are not taken into account since the CSS Stats tool [MJO14] used to download the style sheets, as mentioned above, has pretty-printed them all uniformly.

Figure 4 shows the percentage of violations per coding convention for the complete sample set. The most violated coding convention is the disallowing of the ID selector. ID selectors are disallowed since those should be unique, pointing to only a single element. By using ID selectors, developers limit themselves to styling only a single element, losing the benefit CSS provides regarding to the reuse of styles. However, as can be seen in the graph, not all style sheets adhere to this coding convention. Of all 50 websites, 15 of those have a minimum of

Figure 4: Percentage of violations per coding convention



10% ID selectors, even ranging up to 36.05% (bing.com). Furthermore we have analysed that the `!important` modifier is used on average 16 times every 1000 declarations. Some websites even have more than 5% of all their declarations use the `!important` modifier, with Whatsapp.com being on top with 9.09%. Such a high usage of `!important` modifiers demonstrates bad use/understanding of the cascading characteristic of CSS. Figure 2c shows more information on the usage of the `!important` modifier.

Relating the amount of violations per category of coding conventions to the amount of lines of code in the style sheet, presented some insights in the occurrences of violations. Both the maintainability and compatibility related coding smells showed a strong positive correlation against the amount of lines of code in a style sheet, with their correlations being 0.9938, and 0.9960 respectively. The possible error category also had a positive correlation, being 0.8319. For the performance category there was no significant correlation, as its value was 0.0532. These values are based on only 2–3 coding conventions per category, therefore not being a complete representation for each category, as only a small section of the available coding conventions per category have been used. However, they do indicate that there is a need for better CSS standards to prevent, better CSS analysis tools to detect, and better CSS refactoring tools tools to fix, the decline of quality in style sheets.

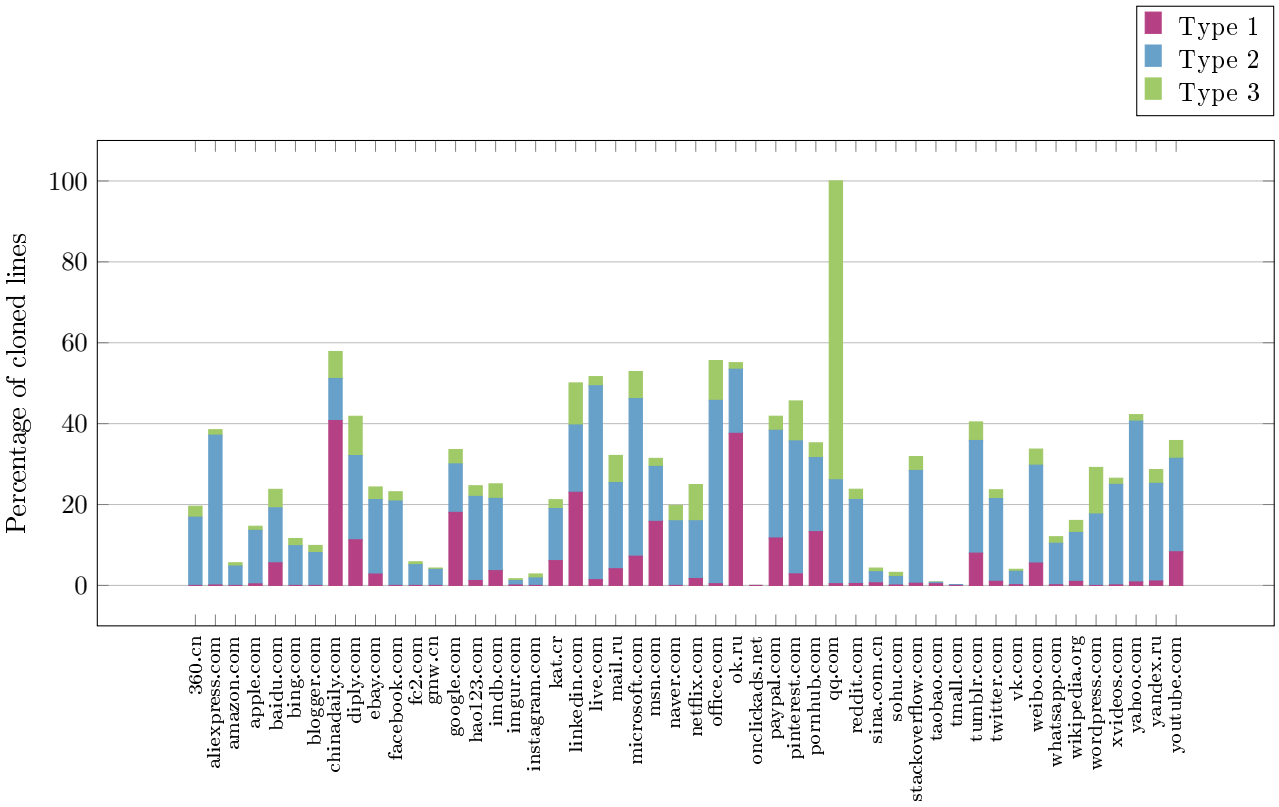
4 Case Study: Detecting Code Clones

Clone analysis, detection, management and tool evaluation have been very active topics in software engineering research at least since 1994, with numbers of papers dedicated to them climbing each year [RZK14]. Clones are usually considered harmful, since they bloat the codebase and hamper proper maintenance since each bug fixed in cloned code, needs its fixes propagated to all code incarnations, including those that significantly evolved since the cloning time. Without joining the ongoing discussion on the usefulness of clones for solving some tasks (in particular in product line implementation), we can point out that in general having or lacking duplicates is a remarkable property of the source code, and quite characteristic of the programming style, partly dictated by the chosen software language. Hence, we are interested in investigating clones in CSS. The expectation is to have results similar to other small DSLs [TC11].

Results of running the clone detector on the sample set, can be seen in Figure 5. These are the results using the current configuration as shown in the list below. Running the clone detector with different configurations would result in different results, however, for the current sample set these specific configurations result in accurate clones.

- Clones should have a minimum mass of 6.
- Clones should occupy a minimum of 3 LOC.
- The following is normalised for type 2 and type 3 clones:
  - File names of style sheets
  - Selectors of rule sets
  - Media queries in @media rules

Figure 5: Percentage of cloned lines per website



  - Names of @counter-style rules
  - Media queries in @import rules
  - Pseudo classes for @page rules

- Type 3 clones should be at least 80% equal.

In Figure 2d, three box plots are shown, one for each clone type. It shows that clones of type 1 have an average length in lines of 4.76% and a median of 0.75%, however more interesting are the outliers which have more than 20%, and for 2 websites even around 40% clones lines. These two are chinadaily.com and ok.ru. A reason for chinadaily.com to have such a high percentage of type 1 clones, is that the style sheet contains a `@media` rule, which aims at screens with a max-width of 1154 pixels. However, instead of only adding rule sets to the `@media` rule that override the previously defined properties for specific resolutions (e.g., for smartphones), it also contains direct duplicates of rule sets from outside the `@media` rule which add no benefit whatsoever. Refactoring the style sheet to remove the specific `@media` rule, and pretty printing the CSS to run it through the clone detector again would be a simple and easy way to verify this, however, the `@media` rule is never closed with a right brace, making it impossible to do, as we can only guess where the `@media` rule should have been closed.

For ok.ru, the high amount of type 1 clones does not seem to be related to any `@media` as was the case for chinadaily.com. For some reason a lot of rule sets that are defined relatively early in the style sheet (first 10,000 lines), are defined again later on in the style sheet (from about 25,000 lines). It seems that the ok.ru website loads 3 style sheets, with 2 of them being fairly equal, containing a lot of the same rule sets. It seems like one of the style sheets is simply duplicated and then partially modified, while not keeping in mind that most rule sets are already defined elsewhere.

Type 2 clones are found the most often, having an average of 17.30% and a median of 16.76%, with one outlier of 38.96% which is microsoft.com. When looking at the style sheet of microsoft.com, it seems like they have used a tool to generate CSS with, maybe SASS or LESS, because the first 2000 lines mostly contains rule sets as shown in Listing 1, rule sets with one or multiple selectors and only a single `width` declaration width a percentage value. Most of these `width` declarations with equal values occurred multiple times in different rule sets, which could explain the high amount of type 2 clones.

Listing 1: Part of the microsoft.com style sheet

```
.CSPvNext .margin-row-fluid>.bp2-col-10-3 {
  width: 27.4%
}
.CSPvNext .margin-row-fluid>.bp2-col-10-4 {
  width: 37.2%
}
.CSPvNext .margin-row-fluid>.bp2-col-10-6 {
  width: 56.8%
}
```

Then there are the type 3 clones, that with the current configurations, result in an average of 4.79% and a median of 2.59%. There is one outlier that stands out the most, as it has 73.91% of type 3 clones. This is the qq.com website, and what is surprising about qq.com is that when combining its type 1, type 2, and type 3 clones, it shows that a 100% of the lines are considered cloned lines. This means that every line in the style sheet, is part of one or more clones. After analysing the qq.com style sheet, the high amount of type 3 clone seems to be a result of copy and pasting. To give an example, in Listing 2 two rule sets are shown taken from the qq.com style sheet. The only thing which sets these rule sets apart are their selectors and the `font-size` and `display` declarations in the second rule set, with the remaining 8 declarations being identical. The two rule sets where not even a 100 lines apart from each other in the style sheet, giving the impression that the developers do not fully understand the inheritance and cascading characteristics of CSS, or that maybe they did, but just wanted to develop the style sheet in a short amount of time while not being bothered by CSS' inheritance and cascading characteristics. Nevertheless, it shows that the 5,449 lines of code that the qq.com style sheet now uses to style its website with, can be reduced significantly.

Listing 2: Part of the qq.com style sheet

```
.nav .erweima {
  width: 103px;
  height: 145px;
  border: 1px solid #dbdbdb;
  padding: 0;
  position: absolute;
  top: 0;
  right: -115px;
  background: url(http://mat1.gtimg.com/www/images/qq2012/erweimaNews1.1.png)
      no-repeat;
}

.navBeta .erweima {
  display: none;
  width: 103px;
  height: 145px;
  border: 1px solid #dbdbdb;
  padding: 0;
  font-size: 12px;
  position: absolute;
  top: 0;
  right: -115px;
  background: url(http://mat1.gtimg.com/www/images/qq2012/erweimaNews1.1.png)
      no-repeat;
}
```

5 Preliminary Conclusions and Future Work

In this report, we have explained how we composed a corpus of realistic CSS code from popular websites, as a part of the effort to build a framework for CSS analysis. We have briefly gone through two case studies that showed how the corpus can be (re)used. The project is still a work in progress, but the corpus is ready and is already serving us well.

Ultimately we will use this corpus of CSS files to compare our framework with existing alternatives, by implementing the same algorithms within various frameworks. Smell detection, clone management, metrics calculation and detecting refactoring opportunities will remain the main themes. Analysing the corpus statistically to see which language features of CSS are more widely used and therefore more crucial to support, is also an interesting option.

Acknowledgements

This report is based on the extended abstract of the presentation given at the SATToSE symposium in Bergen, Norway, on 12 July 2016 [dG16b], as well as on the graduate thesis defended at the University of Amsterdam, The Netherlands, on 21 July 2016 [dG16a].

References

[AMIO12] Adewole Adewumi, Sanjay Misra, and Nicholas Ikhu-Omoregbe. Complexity Metrics for Cascading Style Sheets. In Beniamino Murgante, Osvaldo Gervasi, Sanjay Misra, Nadia Nedjah, Ana Maria A. C. Rocha, David Taniar, and Bernady O. Apduhan, editors, *Proceedings of the 12th International Conference on Computational Science and Its Applications (ICCSA)*, pages 248–257. Springer, 2012. doi:10.1007/978-3-642-31128-4_18.

[BÇHL11] Bert Bos, Tantek Çelik, Ian Hickson, and Håkon Wium Lie. Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. *W3C Recommendation*, June 2011. http://www.w3.org/TR/2011/REC-CSS2-20110607.

[BD16] M. Serdar Biçer and Banu Diri. Defect Prediction for Cascading Style Sheets. *Applied Soft Computing*, 2016. doi:http://dx.doi.org/10.1016/j.asoc.2016.05.038.

[CTB+03] Jordi Cabot, Massimo Tisi, Hugo Brunelière, et al. AtlantEcore Metamodel Zoo. http://www.emn.fr/z-info/atlanmod/index.php/Ecore, 2003.

[CWE06] Hampton Catlin, Natalie Weizenbaum, and Chris Eppstein. SASS: Syntactically Awesome Style Sheets, 2006. http://sass-lang.com.

[dG16a] Nico de Groot. Analysing and Manipulating CSS using the M^3 Model. Master's thesis, Universiteit van Amsterdam, The Netherlands, July 2016. URL: http://www.scriptiesonline.uba.uva.nl/en/scriptie/613750.

[dG16b] Nico de Groot. Analysing CSS using the M3 Model. In *Pre-proceedings of the Ninth Seminar on Advanced Techniques and Tools for Software Evolution (SATToSE)*, 2016. URL: http://sattose.wdfiles.com/local--files/2016:alltalks/SATTOSE2016_paper_10.pdf.

[eEG+11] Tantek Çelik, Elika J. Etemad, Daniel Glazman, Ian Hickson, Peter Linss, and John Williams. Cascading Style Sheets (CSS) Selectors Level 3. *W3C Recommendation*, September 2011. http://www.w3.org/TR/2011/REC-css3-selectors-20110929/.

[Gha14] Golnaz Gharachorlu. Code Smells in Cascading Style Sheets: An Empirical Study and a Predictive Model. Master's thesis, University of British Columbia, Canada, 2014. URL: http://hdl.handle.net/2429/51364.

[GZ16a] Boryana Goncharenko and Vadim Zaytsev. Language Design and Implementation for the Domain of Coding Conventions. In Tijs van der Storm, Emilie Balland, and Dániel Varró, editors, *Proceedings of the Ninth International Conference on Software Language Engineering (SLE)*, pages 90–104, 2016. doi:10.1145/2997364.2997386.

[GZ16b] Boryana Goncharenko and Vadim Zaytsev. Reverse Engineering a CSS Coding Conventions Catalogue. Draft, https://github.com/boryanagoncharenko/CssCoco/blob/master/analysis.md, 2016.

[Maz14] Davood Mazinanian. Dataset for FSE'14 submission, 2014. URL: http://users.encs.concordia.ca/~d_mazina/papers/FSE'14/.

[MJO14] Adam Morse, Brent Jackson, and John Otander. CSS Stats, 2014. http://cssstats.com.

[MTM14] Davood Mazinanian, Nikolaos Tsantalis, and Ali Mesbah. Discovering Refactoring Opportunities in Cascading Style Sheets. In *Proceedings of the 22nd Symposium on the Foundations of Software Engineering (FSE)*, pages 496–506. ACM, 2014. doi:10.1145/2635868.2635879.

[PVZ16a] Leonard Punt, Sjoerd Visscher, and Vadim Zaytsev. Experimental Data for the A?B*A Pattern in CSS: Inputs and Outputs. In *Proceedings of the 32nd International Conference on Software Maintenance and Evolution (ICSME)*, page 616, 2016. Best Artefact Award. doi:10.1109/ICSME.2016.91.

[PVZ16b] Leonard Punt, Sjoerd Visscher, and Vadim Zaytsev. The A?B*A Pattern: Undoing Style in CSS and Refactoring Opportunities it Presents. In *Proceedings of the 32nd International Conference on Software Maintenance and Evolution (ICSME)*, pages 67–77, 2016. doi:10.1109/ICSME.2016.73.

[RZK14] Chanchal K. Roy, Minhaz F. Zibran, and Rainer Koschke. The vision of software clone management: Past, present, and future (Keynote paper). In Serge Demeyer, David Binkley, and Filippo Ricca, editors, *Proceedings of the Software Evolution Week: Conference on Software Maintenance, Reengineering, and Reverse Engineering*, pages 18–33. IEEE Computer Society, 2014. doi:10.1109/CSMR-WCRE.2014.6747168.

[Sch14] Daniel Schauenberg. Development, Deployment & Collaboration at Etsy. In *QCon London*, 2014. https://qconlondon.com/london-2014/london-2014/presentation/Development,%20Deployment%20&%20Collaboration%20at%20Etsy.html.

[SSP+09] Alexis Sellier, Jon Schlinkert, Luke Page, Marcus Bointon, Mária Jurčovičová, Matthew Dean, and Max Mikhailov. Less, 2009. http://lesscss.org.

[TAD+10] Ewan Tempero, Craig Anslow, Jens Dietrich, Ted Han, Jing Li, Markus Lumpe, Hayden Melton, and James Noble. Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies. In *Asia Pacific Software Engineering Conference (APSEC 2010)*, pages 336–345, December 2010.

[TC11] Robert Tairas and Jordi Cabot. Cloning in DSLs: Experiments with OCL. In Anthony M. Sloane and Uwe Aßmann, editors, *Revised Selected Papers of the Fourth International Conference on Software Language Engineering*, volume 6940 of *LNCS*, pages 60–76. Springer, 2011. doi:10.1007/978-3-642-28830-2_4.

[Zak11] Nicholas C. Zakas. Disallow !important, 2011. https://github.com/CSSLint/csslint/wiki/Disallow-!important.

[Zay15] Vadim Zaytsev. Grammar Zoo: A Corpus of Experimental Grammarware. *Fifth Special issue on Experimental Software and Toolkits of Science of Computer Programming (SCP EST5)*, 98:28–51, February 2015. doi:10.1016/j.scico.2014.07.010.