Refactoring and Migration of Cascading Style Sheets

Towards Optimization and Improved Maintainability

Davood Mazinanian
Department of Computer Science and Software Engineering
Concordia University, Montréal, Canada
d_mazina@cse.concordia.ca

ABSTRACT

Cascading Style Sheets is the standard styling language, and is extensively used for defining the presentation of web, mobile and desktop applications. Despite its popularity, the language's design shortcomings have made CSS development and maintenance challenging. This thesis aims at developing techniques for safely transforming CSS code (through refactoring, or migration to a *preprocessor language*), with the goal of optimization and improved maintainability.

CCS Concepts

•**Software and its engineering → Software maintenance tools; Maintaining software;**

Keywords

Cascading style sheets, refactoring, duplication, migration

1. INTRODUCTION

Background. The Lingua Franca of *styling* is Cascading Style Sheets (CSS). On web, CSS allows separating the *presentation* concern from the concerns of organizing content (done in HTML) and interacting with users (fulfilled in JavaScript). CSS allows a consistent content presentation across different media (e.g., displays with various sizes, printers). Today, 91% of top 10 million websites in the Alexa ranking [25] and over 90% of web developers [23] use CSS.

Indeed, CSS now plays a vital role in businesses by directly affecting the perceived user experience. At Dropbox, for example, there existed around 1200 Style Sheet files (exceeding 150K LOC). In one incident, a change in some of these files unknowingly broke the presentation of a revenue-generating page that developers were not aware it even existed, and lack of adequate tools yielded to concealing the fact that the page used the modified CSS file [7]. This could damage Dropbox's professional relationship with the business partner that depended on the broken page.

This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive version was published in the following publication:

FSE'16, November 13–18, 2016, Seattle, WA, USA
ACM. 978-1-4503-4218-6/16/11...
http://dx.doi.org/10.1145/2950290.2983943

Challenges. CSS files contain several *style rules*, in which a *selector* defines the elements that should be styled (e.g., `p` for selecting all paragraphs), and the style of the selected elements are defined using *style declarations* [1]. Despite this apparent simplicity, developing and maintaining CSS code can be challenging [12, 9, 20], primarily because CSS was initially designed without maintainability concerns in mind. For instance, there is no notion of *functions* in CSS; consequently, duplicated code (i.e., clones) is prevalent in CSS [19]. Advanced features like *cascading* and *value propagation* [14] also add to this intricacy. Tool support (e.g., debugging, automatic refactoring) is also quite immature; mainly because CSS code interacts with HTML, which could be generated at server-side, or manipulated by JavaScript at runtime. Also, CSS code can be intertwined with HTML and JavaScript, making static analysis tricky. Consequently, an imprudent refactoring transformation may break the behavior (i.e., the *presentation semantics*) of CSS code.

The tackled problem. Code duplication is considered as a potentially serious problem that might have negative impact on the maintainability and error-proneness of the code base [15, 11]. To our knowledge, no studies had specifically investigated clones in CSS code. Our investigation showed that, on average, 60% of style declarations in the CSS code of modern web applications are duplicated [19]. This may impose a considerable amount of maintenance burden on developers. In addition, duplication leads to larger CSS files, causing extra bandwidth usage, energy consumption and delays in rendering user interfaces. Thus, refactoring might be a resort for reducing duplication in CSS files.

Unfortunately, CSS has very limited support for abstracting duplicated style declarations; e.g., declarations with differences in style values cannot be unified in CSS. To mitigate this limitation, industry has come up with CSS preprocessors (e.g., Less, Sass): super-set languages for CSS that bring the missing features (e.g., functions) to CSS [17]. Using CSS preprocessors is a trend in the industry [6], and some of the aforementioned problems can be solved with them. For instance, duplicated CSS code might be abstracted using functions in a preprocessor language.

Our goal. This thesis is dedicated to the problem of duplication in CSS. We attempt to propose techniques for detecting refactorable duplicated style declarations in CSS, and safely refactoring them, either directly in the CSS code, or by migrating existing CSS code to a preprocessor language. We propose methods for testing the correctness of the applied refactorings, and evaluate it on a large data set of web applications. A tool suite is under development that automates the application of transformations right in the IDE.

Finally, the relevance of the identified refactoring/migration opportunities will be evaluated through a user study.

Related work. Despite the widespread popularity, CSS is utterly underrated from academia. The few studies on CSS maintenance are limited to finding dead code [20, 9, 2], detecting code smells [10], defining quality metrics [12], and visualizing change impact [13]. As mentioned, we investigated clones in CSS, with the goal of optimizing the size of CSS files [19]. Having the same goal, Boash et al. proposed techniques to *reason* about the selectors and declarations that can be safely removed from style sheets [4, 2, 3].

Expected contributions. This research can gain the attention of software engineering community to improve the state-of-the-art techniques in maintaining CSS code. The developed infrastructure for analyzing CSS code of web applications and applying transformations can be used by other researchers to this aim. In addition, the developed techniques in this work, and the corresponding tool support, can facilitate the daily work of countless developers that deal with CSS code, e.g., front-end web developers.

2. THE STATE OF THE RESEARCH

Refactoring duplication in CSS. In a previous study [19], we defined three types of duplicated declarations, i.e., declarations that result to identical *styles* (e.g., they all make a hyperlink blue). We found that, in the CSS files of our dataset (containing 38 popular websites), around 60% of declarations fell in one of the mentioned duplication types. We defined some preconditions which, when they are met, one can safely refactor the duplicated declarations. On average, our approach identified 165 refactoring opportunities in each CSS file, of which 62 could be safely applied. Applying refactorings yielded up to 40% file size reduction.

Refactoring using this technique is not always possible; because it includes re-ordering of CSS style rules, which could result in breaking an existing *order dependency* between style rules [19]. Finding these dependencies is not trivial since, for instance, CSS code can be affecting HTML pages which are manipulated by JavaScript at runtime. Thus, we used Crawljax [21], a tool that allows crawling web applications that heavily use JavaScript, and extracted order dependencies using a hybrid (static and dynamic) approach.

Empirical study on the use of CSS preprocessors. When two style declarations have the same properties but neither equal nor equivalent values, they can be only unified by *parameterizing* the differences in a function-like construct; something which is missing in CSS. In CSS preprocessors, *mixins* fill this gap [17]. To identify relevant opportunities for extracting mixins from duplicated CSS code, we need to study how developers usually use them, so that our recommendations are closer to what developers would manually extract. We studied the source code of 150 websites, having their CSS code written in two popular preprocessors, and found out (among other findings) that, developers tend to define mixins with less than three declarations, having one parameter on average, mostly grouping declarations that style the same property for different web browsers [17].

Automatic migration of CSS code to preprocessors. With the gained knowledge in the previous work, we have devised a technique that finds duplicated declarations in the CSS code, parameterizes differences, extracts them into a mixin, and tests whether these changes are safe (i.e., the technique *migrates* existing CSS code to take advantage of mixins by removing duplications) [18]. For finding duplications, we used a relaxed version of the approach used in our first work [19], so that it allows differences in style values. These differences are identified and parameterized, by mapping corresponding values in the duplicated declarations. We use an approach that mimics the behavior of web browsers, so that the style values with the same *role* are mapped together (e.g., style values that both define the *width* of a border are mapped together). For testing, we compare the applied styles before and after refactoring for each of the elements of target documents. Our evaluation shows that our approach is able to propose opportunities that are safe to apply, with 98% recall (i.e., it detects a large portion of mixins manually defined by developers) [18].

3. OPEN ISSUES AND FUTURE WORK

Other types of migration. Besides mixins, migrating to preprocessors can embrace other types of transformations, e.g., introducing *variables*, *nested rules*, or the *extend* construct [17]. We are going to explore approaches for migrating CSS to take advantage of these constructs in preprocessor languages, and include them in our migration tool suite.

Ranking migration opportunities. Our technique for migrating CSS to preprocessors using mixins [18] suffers from the limitation that it may suggest a large number of opportunities to the developer, due to the abundance of duplicated declarations. This could happen when suggesting opportunities for extracting other constructs (e.g., variables). As a result, we are going to devise and evaluate ranking mechanisms, so that the tool ranks higher the opportunities that are more *relevant*. For instance, as developers tend to create mixins for grouping declarations which style the same property in different web browsers (for achieving a consistent presentation across them) [17], opportunities extracted from such properties should be ranked higher. Likewise, we aim at finding similar patterns, by investigating existing preprocessor code bases, or using the *context* information (i.e., HTML pages on which the CSS code under migration is applied), or interviewing developers, to extract features that may help in ranking. We will then explore various techniques (e.g., statistical models) for feature selection and ranking.

Testing. The introduced testing technique [18] is far from a complete testing solution for CSS. The state-of-the-art techniques for detecting presentational differences in HTML pages across different web browsers [5, 16, 24] might be exploited for determining whether the transformations are safe, however, these approaches are computationally expensive, and suffer from false positives. As a result, we need a technique for enabling developers to write tests for CSS code, so that it can be tested after refactoring/migration. We might also be able to aid developers by automatically generating test cases. This can be done by extending our testing technique [18], by taking advantage of the latest advances in testing JavaScript [22, 8], which can facilitate understanding the interaction between CSS and HTML.

Other possible directions. CSS is extensively used, yet extremely under-researched. Thus, there are several interesting research topics to investigate, including but not limited to: refactoring CSS to eliminate code smells (e.g., the ones proposed in [10]), empirical studies on how CSS code is maintained and evolved, or how bugs look like and fixed in CSS code (e.g., performance bugs versus styling bugs).

4. REFERENCES

[1] Css syntax module level 3. Technical report, World Wide Web Consortium, November 2013.

[2] M. Bosch, P. Genevès, and N. Layaïda. Automated refactoring for size reduction of css style sheets. In *Proceedings of the 2014 ACM Symposium on Document Engineering (DocEng)*, pages 13–16, 2014.

[3] M. Bosch, P. Genevès, and N. Layaïda. Reasoning with style. In *Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI)*, pages 2227–2233, 2015.

[4] M. Bosch, P. Genevès, and N. Layaïda. Automated and Semantics-Preserving CSS Refactoring. Technical report, HAL - Inria Open Archive, Nov. 2014.

[5] S. R. Choudhary, H. Versee, and A. Orso. WEBDIFF: Automated identification of cross-browser issues in web applications. In *Proceedings of the 26th IEEE International Conference on Software Maintenance (ICSM)*, pages 1–10, 2010.

[6] C. Coyier. Popularity of CSS Preprocessors. http://css-tricks.com/poll-results-popularity-of-css-preprocessors/.

[7] D. Eden. Move slow and fix things. http://www.thedotpost.com/2015/12/daniel-eden-move-slow-and-fix-things, 2015. Talk at dotCSS Conference.

[8] A. M. Fard, A. Mesbah, and E. Wohlstadter. Generating Fixtures for JavaScript Unit Testing. In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 190–200, 2015.

[9] P. Genevès, N. Layaïda, and V. Quint. On the analysis of cascading style sheets. In *Proceedings of the 21st International Conference on World Wide Web (WWW)*, pages 809–818, 2012.

[10] G. Gharachorlu. *Code smells in Cascading Style Sheets: an empirical study and a predictive model.* Master's thesis, University of British Columbia, 2014.

[11] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner. Do code clones matter? In *Proceedings of the 31st International Conference on Software Engineering (ICSE)*, pages 485–495, 2009.

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

[13] H.-S. Liang, K.-H. Kuo, P.-W. Lee, Y.-C. Chan, Y.-C. Lin, and M. Y. Chen. SeeSS: Seeing What I Broke – Visualizing Change Impact of Cascading Style Sheets (CSS). In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, pages 353–356, 2013.

[14] H. W. Lie. *Cascading Style Sheets.* Ph.D. Thesis, University of Oslo, Norway, 2005.

[15] A. Lozano and M. Wermelinger. Assessing the effect of clones on changeability. In *Proceedings of the 24th IEEE International Conference on Software Maintenance (ICSM)*, pages 227–236, 2008.

[16] S. Mahajan and W. G. J. Halfond. WebSee: A Tool for Debugging HTML Presentation Failures. In *Proceedings of the 8th International Conference on Software Testing, Verification and Validation (ICST)*, pages 1–8, 2015.

[17] D. Mazinanian and N. Tsantalis. An empirical study on the use of CSS preprocessors. In *Proceedings of the 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2016.

[18] D. Mazinanian and N. Tsantalis. Migrating Cascading Style Sheets to Preprocessors by Introducing Mixins. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pages 672–683, 2016.

[19] D. Mazinanian, N. Tsantalis, and A. Mesbah. Discovering Refactoring Opportunities in Cascading Style Sheets. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, pages 496–506, 2014.

[20] A. Mesbah and S. Mirshokraie. Automated analysis of CSS rules to support style maintenance. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 408–418, 2012.

[21] 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*, 6(1):3:1–3:30, 2012.

[22] S. Mirshokraie, A. Mesbah, and K. Pattabiraman. JSEFT: Automated Javascript Unit Test Generation. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation*, pages 1–10, 2015.

[23] Mozilla Developer Network. Web developer survey research. Technical report, Mozilla, 2010.

[24] S. Roy Choudhary, M. R. Prasad, and A. Orso. X-PERT: Accurate Identification of Cross-browser Issues in Web Applications. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE)*, pages 702–711, 2013.

[25] W3Techs. World Wide Web Technology Surveys. http://w3techs.com/technologies/details/ce-css/all/all.