DHBW Mosbach

Cascading Style Sheets - Structural Problems and Solutions

Term paper of the third academic year

Study course:	Onlinemedien
Class:	ON13
Author:	Wirth, Jan
Date of birth:	08.09.1995
Place of Birth:	Eberbach
Immatriculation number:	5445439
Scientific tutor:	Prof. Dr. Mester, Arnulf
Name and domicile of training company:	visual4 GmbH, Stuttgart

Submission date:	23.11.2015
Signature:	_____

Contents

Listings

1 Introduction

In 2012 the online magazine CSS-Tricks conducted a poll on cascading style sheets (CSS) preprocessors. The summary of the 13000 responses given shows that most people that tried preprocessor syntax prefer it.[1] We consider this as a symptom for language specific problems of vanilla (standard) CSS that arise in the development of a modern web user interface (UI).

The following issues were considered before setting the problem scope of this paper: Stylesheet size and structure do have little perceptible impact on both load time and rendering performance.[2,3] Browser inconsistencies have a relatively straightforward solution, namely normalize or reset style sheets.[4,5] Solutions to layout problems like collapsing margins or unexpected float behavior tend to be less straightforward, but are rarely debated.[6,7] Artifacts of the search for effective and efficient methods to structure CSS can be observed back to the year 2003.[8]

In the recent years several different practices and sets of practices to structure style sheets have been promoted by widely recognized companies and individuals of the web industry. Among others, those include Yandex[9] and Yahoo[10]. In defiance of the abundance of recommended methods and promoted best practices, comparative information based on problem-orientend examination is

[1] cf. Chris Coyier. *Poll Results: Popularity of CSS Preprocessors | CSS-Tricks*. https://css-tricks.com/poll-results-popularity-of-css-preprocessors/. (Visited on 02/11/2016)

[2] cf. Steve Sounders. *Performance Impact of CSS Selectors | High Performance Web Sites*. http://www.stevesouders.com/blog/2009/03/10/performance-impact-of-css-selectors/. (Visited on 01/23/2016)

[3] cf. Tab Jr. Atkins. *Re: Multiple assignment structure (suggestion) on 2014-11-27 (www-style@w3.org from November 2014)*. http://lists.w3.org/Archives/Public/www-style/2014Nov/0568.html. (Visited on 02/14/2016). Nov. 2014

[4] cf. Eric A. Meyer. *CSS Tools: Reset CSS*. http://meyerweb.com/eric/tools/css/reset/. (Visited on 02/14/2016)

[5] cf. Nicolas Gallagher. *About normalize.css – Nicolas Gallagher*. http://nicolasgallagher.com/about-normalize-css/. (Visited on 02/14/2016). 2

[6] cf. Eric A. Meyer. *Containing Floats (Complex Spiral Consulting)*. http://www.complexspiral.com/publications/containing-floats/. (Visited on 02/14/2016). Aug. 2003

[7] cf. Eric A. Meyer. *Uncollapsing Margins (Complex Spiral Consulting)*. http://www.complexspiral.com/publications/uncollapsing-margins/. (Visited on 02/14/2016). Nov. 2004

[8] cf. *[css-d] CSS Methodologies*. http://archivist.incutio.com/viewlist/css-discuss/28720. (Visited on 01/18/2016)

[9] cf. *BEM: Key concepts / Methodology / BEM. Block, Element, Modifier / BEM*. https://en.bem.info/method/key-concepts/. (Visited on 02/09/2016). Yandex

[10] cf. *Atomic CSS*. http://acss.io/. (Visited on 02/11/2016). Yahoo

scarce. We will analyse common structural problems of CSS. Consecutively, solutions for those problems will be examined. The scope is limited to vanilla CSS solutions but relevant tools like preprocessors and template engines are noted as well. This is because these tools are additional technologies that do not tackle the problems at their roots and are possibly limited in applicability. The aim of this paper is to provide comprehensible insight for frontend developers that want to challenge best practices.

The following conventions are used in this paper: Three question marks `???`signify the omission of an arbitrarily sized part of a listing or drawing. In code examples, the document `index.html` has a reference to the stylesheet `style.css`. Code comments in the format of `\\ index.html` define which file the following lines of code belong to.

2 Structural Problems of CSS

In this chapter, we identify structural problems of CSS and explain negative effects with the help of code samples. Consecutively we present solutions for those problems. The structural problems are selected by the expected amount of technical debt their encounter adds to a project. They are a sub-set of the problems that practices sets mentioned in chapter 1 are addressing and belong in either of two categories. Violations of the Don't Repeat Yourself (DRY) principle are common culprits in software development but inherent to CSS because CSS itself provides no means to store and recall values. This includes, but is not limited to, selectors, attribute-value-pairs and entire rules. Now we examine leaks as a cause for unexpected side-effects.

2.1 Scope Leaks

CSS were created with the software design principle Separation of Concerns (SoC) and the intent to separate content and visual styling.[11] In contrast to inline styles and embedded style sheets they are reusable across different Hypertext Markup Language (HTML) documents. The rules of a stylesheet referenced in an HTML-document may apply to any part of the DOM if selectors match. This is because the scope of CSS rules is global by default. We also call them 'unscoped'. When we consider a stylesheet as a program that is executed in the context of the Document Object Model (DOM), the individual CSS-rules behave like as impure functions executed in given order. This means they alter their arguments.[12]

Similar to a program with impure functions, style sheets with of unscoped styles can have unexpected side-effects. For example, when we add or modify a rule with the intention to modify the visual appearance of a specific element of the DOM, the style change can apply to other elements as well. This so called 'scope leak'[13] can occur in any HTML-Document that references this very stylesheet. When the developer undertaking the changes is not aware of all applications of the rule he changes, some of resulting effects are to be considered unforeseen.

[11] cf. Håkon Wium Lie. *CSS*. Mar. 2005 p.74

[12] cf. Yves Lafont. "The linear abstract machine". In: *Theoretical computer science* 59.1-2 (1988), pp. 157–180 p.158

[13] cf. Jeff Escalante. *MPG CSS*. `http://markup.im/#rc6E3Rdg`. (Visited on 02/03/2016)

The following drawing visualises the structrue of a hypothetical DOM :

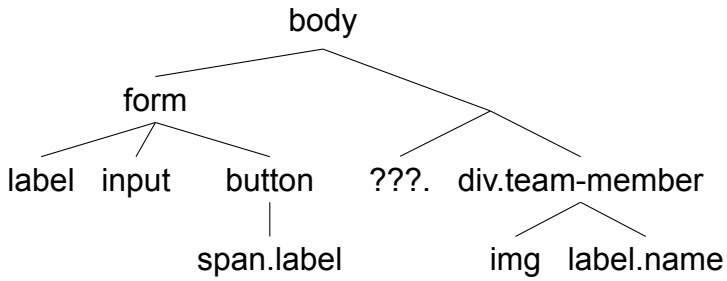

Figure 2.1: Example DOM tree

This rule was written with the intention to set the type of the label inside the form in bold weight:

Listing 2.1: Rule with side effects

```
1 label {
2   font-weight: bold;
3 }
```

Applying these rules achieves the intended effect, but also changes the weight of the type the name of a team member is set in. The `???` denote an arbitrarily complex part of the DOM. Depending on the complexity of the DOM and the tools at hand, the costs connected to locating the effects of a modification may be high. The cascade and specificity mechanism of CSS may add some complexity to side effects because individual rules have the potential to interact with each other. We assume the presence of the following, more specific, unscoped rule:

Listing 2.2: Rule with non-generic specificity

```
1 .label{
2   font-weight: light;
3 }
```

Now the first rule does not take effect because the class selector of the second rule is more specific.[14] To achieve the targeted visual appearance, further modifications are necessary, e.g. increasing the specificity of the new rule, which again results in an increased complexity of the source code.

The risk of unforeseen side effects in the development of generally unscoped CSS is increased with the complexity of the codebase and the number of developers involved. Scope leaks increase the cost with comprehending the source code,

[14] cf. *Selectors Level 3*. `https://www.w3.org/TR/css3-selectors/#specificity`. (Visited on 02/14/2016). W3C

i.e. to acquire and maintain knowledge of the system that enables a developer to modify it without unexpected side-effects while achieving the desired effect. We can draw a parallel between non-local rules and Wulf and Shaw's considerations of non-local variables as 'a major contributing factor in programs which are difficult to understand' for similar reasons.[15]

It remains to mention that global CSS-rules have a valid use case as base styles. Base styles provide visual consistency across across multiple HTML documents and use element selectors.[16]

2.2 Violations of the Don't Repeat Yourself Principle

The DRY Principle was enunciated in 2000 by Hunt and Thomas. It states that 'Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.'[17] Many violations of the DRY principle in CSS are directly linked to the lack of variables. In the following listing we can observe what is most likely a violation of DRY

Listing 2.3: Violation of DRY

```
1 button {
2   margin: 10px;
3   ???
4 }
5
6 a {
7   margin: 10px;
8   ???
9 }
```

We assume that the intention of the developer was to give both links and buttons the same margin whose exact value is derived from a grid system. The singular piece of knowledge originating in the grid system now has two authoritative representations in the code. This lowers the maintainability because a single change of the grid dimensions results in two or more necessary of the code. Even automated tools like find and replace are prone to errors of their operators as the

[15] William Wulf and Mary Shaw. "Global variable considered harmful". In: *ACM Sigplan notices* 8.2 (1973), pp. 28–34 p.28

[16] cf. Escalante, *MPG CSS*

[17] David Thomas and Andy Hunt. *The Pragmatic Programmer: From Journeyman to Master*. 1999 p. 27

operation may require detailed knowledge of the target system.[18] To exemplify this, we consider the following listing:

Listing 2.4: Coincidental color definition overlap

```
1 .important {
2     background-color: red;
3 }
4
5 .error {
6     background-color: red;
7 }
```

Here, by coincidence, elements decorated with the classes `error` and `important` are both styled with red background. In common sense, information about an error is important, but not all important information is an error. This means that the two occurences of `red` may change independently because they actually mean different things. If this happens, a simple find and replace operation on all occurences of `red` with the intent to fulfill changed design requirements results in an unexpected side-effect. The bigger a project gets, the more likely are such coincidental overlaps.

2.3 Solutions

After describing structural problems of CSS, we now introduce methods taken from the sets of practices introduced in chapter 1. We note that the methods presented are probably not suitable for every developer and project due to personal preference and established conventions. The methods covered here are introduced and evaluated based on code samples. In the evaluation we will also consider semanticity of selectors. This refers to what a reader can infer about the purpose of a rule by reading the associated selector alone. We note that semantics here are not to be confused with microformats and semantic HTML5-Elements and thus are not relevant to search engine optimisation (SEO).

[18] cf. Raja Parasuraman and Victor Riley. “Humans and automation: Use, misuse, disuse, abuse”. In: *Human Factors: The Journal of the Human Factors and Ergonomics Society* 39.2 (1997), pp. 230–253 p.408

2.3.1 Modularisation

The guidelines of Block Element Modifier (BEM) and Styles and UI Toolkit (SUIT) share the concept of blocks. A block encapsulates HTML and CSS as well as 'other implementation technologies'[19] and are also referred to as components or modules.[20,21] Both guidelines define similar rules for structuring markup, class names and selectors for blocks. The following example displays the construction of component and child-element selectors in SUIT.

Listing 2.5: SUIT naming convention

```
// index.html

<div class='BlockName'>
  <div class='BlockName-descendantName'>
  ???

// style.css

.BlockName {}
.BlockName-descendantName {}
```

BEM Whereas the delimiters can be chosen freely in BEM, the naming conventions are more specific for SUIT. BEM recommends not nesting descendants of a block within each other.[22] The reason for capitalising the block selector is to avoid name collisions with existing code, provided that existing class names begin with a lowercase character.[23]

To this date, while some browsers support scoped `<style>`-tags, the CSS scoping module level 1 is still in draft.[24,25] The World Wide Web Consortium (W3C) draft on CSS scoping defines a syntax that is similar to that of media queries. However, media queries no not operate based on the DOM-structure but are

[19] *BEM: Key concepts / Methodology / BEM. Block, Element, Modifier / BEM*

[20] cf. Cameron Hunter et al. *SUIT CSS: style tools for UI components*. `https://suitcss.github.io/`. (Visited on 02/09/2016). suitcss

[21] cf. *BEM: Key concepts / Methodology / BEM. Block, Element, Modifier / BEM*

[22] cf. *FAQ / BEM. Block, Element, Modifier / BEM*. `https://en.bem.info/faq`. (Visited on 02/15/2016). Yandex

[23] cf. *The differences between BEM and SUIT CSS naming conventions - Stack Overflow*. `http://stackoverflow.com/questions/30892764/the-differences-between-bem-and-suit-css-naming-conventions`. (Visited on 02/05/2016)

[24] cf. *CSS Scoping Module Level 1*. `https://drafts.csswg.org/css-scoping/`. (Visited on 01/24/2016)

[25] cf. *style – style (presentation) information - HTML5*. `http://w3c.github.io/html-reference/style.html`. (Visited on 01/24/2016). W3C

scoping styles based on various properties of the rendering device.[26] The `@scope` allows surrounding a set of rules with a selector, limiting these rules' scope to children of the element matched by that selector.

[26] cf. *Media Queries*. `https://www.w3.org/TR/css3-mediaqueries/`. (Visited on 02/03/2016). W3C

Listing 2.6: Scope module example

```
@scope div {
  span {
  color: blue;
  }
}
```

In terms of limiting the visibility of rules this has the exact same effect as prefixing it with that selector.[27] Atkins, member of the W3C CSS Working Group, describes this syntax as 'selector sugar'.[28]

A selector chain with more than one selector effectively scopes the rule by the first selector:

Listing 2.7: Basic scoping with prefixed selector

```
.sidebar span {
  background: red;
}
.sidebar a {
  color: blue;
}
```

The conventions defined by SUIT and BEM achieve module-scoping differently. Listing 2.5 shows that the scoping here does not utilise the cascade but works through prefixing the actual class names. Block descendant styles are applied to elements decorated with the classname `ComponentName-descendantName`, regardless of their position in the DOM. Escalante criticises this approach. Instead of leveraging the functionality of the cascade, BEM and SUIT annul it to 'replicate it [...] in a less efficient manner'.[29] Inheriting the name of the block into the class name of a descendant causes repetition in both CSS and HTML.[30] *Title CSS* leverages the cascade for scope emmulation.[31] Through separating the block identifier as distinct class, repetition within the HTML-markup is avoided:

Listing 2.8: Title CSS Convention

```
.BlockName {}
.BlockName .descendantName{}
```

[27] cf. *CSS Scoping Module Level 1*

[28] cf. Tab Atkins. *Selector Sugar from Tab Atkins Jr. on 2008-10-09 (www-style@w3.org from October 2008)*. `http://lists.w3.org/Archives/Public/www-style/2008Oct/0066.html`. (Visited on 02/15/2016). Oct. 2008

[29] Escalante, *MPG CSS*

[30] cf. ibid.

[31] cf. Jonathan Cuthbert. *cuth/Title-CSS: A CSS organization methodology*. `https://github.com/cuth/Title-CSS`. (Visited on 02/05/2016)

Besides avoiding repetition, Cuthbert, the creator of Title CSS, adds ease-of-writing as well as readability as rationales for *Title CSS* as naming convention. If Title CSS is executed in the style of the given example, the risk of scope leaks persists. Whereas BEM and SUIT target descendants directly through class names, the cascade may also target identically named descendants of blocks nested within other blocks. To avoid this, we modify the conventions for creating descendant selectors like so:

Listing 2.9: Title CSS Convention modified

```
1 .BlockName > .descendantName{}
```

The > component of the selector ensures that only direct childrent of `BlockName`are matched with `descendantName`.

2.3.2 Page Scoping

Coyier introduced and Escalante recommends decorating the body element of each HTML document of a project with an ID.[32,33] Prefixing a selector with such an ID scopes the associated rule to the correspondent pages. Generally, page-specific styles may be used to define visual cues as orientation aids. A concrete example is distinguishing the header of an article page from that of other pages.

Listing 2.10: Page-specific exception

```
1 header {
2   background: transparent;
3 }
4 #article header {
5   background: black;
6 }
```

Coyier also exemplifies the use of page-specific styles through highlighting the currently selected entry of the main menu without changing the markup of the latter.

[32] cf. Chris Coyier. *ID Your Body For Greater CSS Control and Specificity | CSS-Tricks*. `https://css-tricks.com/id-your-body-for-greater-css-control-and-specificity/`. (Visited on 02/04/2016). Dec. 2007

[33] cf. Escalante, *MPG CSS*

Listing 2.11: Highlighting of navigation entry with page scope

```
// index.html

<body id='about'>
  <nav>
    <ul>
      <li class='nav-item home'>
      <li class='nav-item about'>
    ???


// style.css

#about .nav-item.about {
  font-weight: bold;
}
```

However this technique should be applied with caution. The markup and the styles in this example are coupled tightly and effectively violate the principle of SoC because there is no visual information about the currently selected entry without the stylesheet. Each navigation entry requires an additional rule to function correctly in the context of the UI. Because this leads to bloat in CSS code when the number of navigation entry is growing, a dynamic JavaScript (JS) client- or server-side solution that modifies the markup is a better choice when aiming for scalability. BEM calls IDs that are assigned to the body tags 'global modifiers'. BEM does not accommodate the concept of global modifiers.[34] The rationale behind this convention is not exposed, but it can be accounted to aforementioned caveats. The following section explains how, among other techniques, more granular modifiers can be utilised to reduce repetition.

2.3.3 Decoupling

There are several tools and methods available that allow eliminating repetitions in CSS-code. Variables are, to this date, only fully supported by Firefox Desktop and Android as well as Chrome 49.[35] Besides variables there are no CSS to eliminate repetition within style sheets alone.

[34] cf. *FAQ / BEM. Block, Element, Modifier / BEM*

[35] cf. *Using CSS variables - CSS | MDN*. `https://developer.mozilla.org/en-US/docs/Web/CSS/Using_CSS_variables#Browser_compatibility`. (Visited on 02/09/2016). Mozilla Foundation

All common CSS-Preprocessors provide a variable feature. These variables are only variable during the actual processing and not at interpretation-time. Thus preprocessor variables are effectively constants.[36] To eliminate the duplication constants are sufficient. Using preprocessors allows eliminating repetition in the source code to some degree but the repetition will still be observable in the processed CSS.

Atomic CSS

An approach to completely eliminate repetition in vanilla CSS is Atomic CSS. It was presentend in 2013 by Koblentz, frontend engineer at Yahoo. It advocates splitting the CSS code in the smallest possible parts. These parts are usually rules with a single attribute-value pair. Atomic CSS rules represent a specific visual appearance. The class names of the individual rules are constructed based on their style attributes. Atomic CSS effectively decouples the CSS from the structure of the HTML document.

Listing 2.12: Atomic CSS Rule

```
1 .Mend-10 {
2     margin-left: 10px;
3 }
```

Rules do not have a specific context and thus are highly reusable.[37] There are multiple tradeoffs to the Atomic CSS approach:

- **Additional classes in markup:** Visual attributes are grouped through class assignment rather than below a specific selector. This means more classes need to be assigned to each element whose visual appearence deviates from default and base styles. The resulting markup may be considered bloated.
- **Presentational class names and no scoping:** Because of the radical decoupling the CSS rules are named by the visual attribute they yield. The lack of scoping possibly impedes the developer's ability to localize the effect of changes.

[36] cf. Jan Wirth. "Analyse und Vergleich der CSS Präprozessoren LESS, Sass und Stylus im Hinblick auf benötigte Einarbeitungszeit, Praxisnutzen und Vor- bzw. Nachteile gegenüber reinem CSS". DHBW Mosbach, Sept. 2014 p.27

[37] cf. Thierry Koblentz. *Challenging CSS Best Practices – Smashing Magazine*. https://www.smashingmagazine.com/2013/10/challenging-css-best-practices-atomic-approach/. (Visited on 02/10/2016). Yahoo

- **Changing names:** The W3C states that 'Good names don't change'.[38] The Usage of class names that inherit attribute values are a violation of DRY. Replacing these inherited attribute values with names like `small`may eliminate the need to change class names when the actual attribute values change.
- **Unused rules:** As there is no semantic link between the elements of the UI and the class names, removing nodes from the DOM may result in orphande rules.
- **State management:** Because Atomic CSS does not accomodate modifiers, states of UI elements have to be defined as a set of atomic classes within the context of the UI element.

The development of the Atomic CSS approach has since been continued by Yahoo but is not examined in detail, because the new methods involve the usage of processors. Current versions do not involve direct editing of CSS-files. The class names assigned to the DOM define the actual style attributes. Atomizer, a build tool that extracts CSS rules from the definitions in the markup is used to generate CSS-files. Atomizer supports the usage of variables. Whereas an overhead of unused rules in the CSS-files is prevented by this approach, the markup is still studded with non-semantically named classes.[39]

Inheritance

Whereas Atomic CSS recommends a quite radical approach, the methods described in subsection 2.3.1 can be extended to emulate patterns from object oriented programming.

According to SOLID[40], CSS classes should be modeled with a single responsibility in mind, e.g. styling buttons.[41] BEM and SUIT use this concept in the development of components. Components can be extended by sub-classes. In the following code listing the `BaseComponent`styles are inherited through appending the selector of the extending class.

[38] *Use class with semantics in mind - Quality Web Tips*. `https://www.w3.org/QA/Tips/goodclassnames`. (Visited on 02/10/2016). W3C

[39] cf. *Atomic CSS*

[40] Single Responsibility Principle, Open/Closed Principle, Liskov Substitution Principle, Interface Segregation Principle, Dependency Inversion Principle (SOLID)

[41] Miller H. Borges Medeiros. *SOLID CSS*. `http://blog.millermedeiros.com/solid-css/`. (Visited on 02/05/2016)

Listing 2.13: Module extension

```
.BaseComponent,
.BaseComponent-subclass {}

.BaseComponent-subclass {}

<div class='BaseComponent-subclass'>
```

The sub class may then be assigned to a DOM element to apply both base styles and extended styles to it. This method respects the Open / Closed principle. Base classes should have a single responsibility and thus be impartible. If this is respected, a base class ideally does not change if the requirements to the UI are constant. The base classes are then open to extension but closed to change.

Decorators have a similar effect. Instead of modeling inheritance within the stylesheet, both base and decorator classes are modeled seperately. The decorator is then applied to a DOM element with the according base class. The visual apperance of this element is then modified without altering that of the base class. Decorators shift some repetition to the HTML document in contrast to subclassing:

Listing 2.14: Module decorator

```
.BaseComponent {}
.BaseComponent-decorator {}

<div class='BaseComponent BaseComponent-decorator'>
```

Utility classes

Some styling is hard to attribute to a specific component. This is proven by the fact that in these cases it is hard to find a descendant name that makes sense in the naming convention. Utility classes are used to cover such situations.[42] An common use case is the implementation of workarounds for limitations of CSS and HTML. Similiar to Atomic CSS classes, the names of utility classes are based on their specific styles and have no context. This makes them highly reusable. The following example displays the use of a utility class to prevent single orphaned words which is not possible without additional markup:

[42] cf. Hunter et al., *SUIT CSS: style tools for UI components*

Listing 2.15: Utilities

```
.u-nowrap {
    white-space: nowrap;
}

<h1>
    Lorem Ipsum dolor
    <span class='u-nowrap'> sit amet.</span>
</h1>
```

A class name like `.Heading-nowrap` would break with the naming scheme of modules and limit the reusability of this class. The heading is not any less of a heading of the `.u-nowrap`element is missing. This element is markup that was added with the sole purpose to extend the capabilities of HTML and CSS and may be reused for any texts in the UI. Also, in this example, if a template engine is in place, it makes sense to delegate the wrapping of the last two words to it. Utility classes do not change. They barely yield properties with numeric values but rather those with a limited set of options.

3 Conclusion

CSS as a language is easy to learn and hard to master. The speed at which tangible results can be produced when using CSS is fairly high but may decline quickly because of the structural problems of CSS Because of that, the development process of CSS should implement adequate and effective methods to prevent scope leaks and gratuitous repetition with the aim to improve readability and scalability. Cutting the technical debt caused by structural problems of CSS usually involves not only modifying style sheets but also the HTML-documents that reference them. The cost of this may be increased by tight coupling of other software components with the HTML-component.

The examination of the selected methods showed that there are effective solutions available to prevent scope leaks and confine gratuitious repetition. However, some methods aimed at enforcing the DRY principle are incompatible with each other. They exist on a scale that ranges from complete decoupling and maximum reusability to flexible modularity with limited reusability. The decision about when to apply scoping methods or naming and structure conventions remains to be decided in concrete project context with respect to developer preferences and established conventions.

The list of problems and solutions evaluated in this paper is incomplete. As the selection of problems and solution is based on estimations of negative impact on code structure and the attention attributed to each, it makes sense to examine and measure the impact of other issues like state management, performance, browser inconsistencies etc.. A more complete, measured view of language-specific problems in the development of CSS could enable researchers to design abstract pattern libraries that provide granular coverage for all important issues without defining implementation details. In order to reach the level of knowledge similar to general knowledge in software engineering, more efforts must be put into the research of front-end development, specifically CSS.

Bibliography

Atkins, Tab. *Selector Sugar from Tab Atkins Jr. on 2008-10-09 (www-style@w3.org from October 2008)*. http://lists.w3.org/Archives/Public/www-style/2008Oct/0066.html. (Visited on 02/15/2016). Oct. 2008.

Atkins, Tab Jr. *Re: Multiple assignment structure (suggestion) on 2014-11-27 (www-style@w3.org from November 2014)*. http://lists.w3.org/Archives/Public/www-style/2014Nov/0568.html. (Visited on 02/14/2016). Nov. 2014.

Atomic CSS. http://acss.io/. (Visited on 02/11/2016). Yahoo.

BEM: Key concepts / Methodology / BEM. Block, Element, Modifier / BEM. https://en.bem.info/method/key-concepts/. (Visited on 02/09/2016). Yandex.

Coyier, Chris. *ID Your Body For Greater CSS Control and Specificity | CSS-Tricks*. https://css-tricks.com/id-your-body-for-greater-css-control-and-specificity/. (Visited on 02/04/2016). Dec. 2007.

– *Poll Results: Popularity of CSS Preprocessors | CSS-Tricks*. https://css-tricks.com/poll-results-popularity-of-css-preprocessors/. (Visited on 02/11/2016).

[css-d] CSS Methodologies. http://archivist.incutio.com/viewlist/css-discuss/28720. (Visited on 01/18/2016).

CSS Scoping Module Level 1. https://drafts.csswg.org/css-scoping/. (Visited on 01/24/2016).

Cuthbert, Jonathan. *cuth/Title-CSS: A CSS organization methodology*. https://github.com/cuth/Title-CSS. (Visited on 02/05/2016).

Escalante, Jeff. *MPG CSS*. http://markup.im/#rc6E3Rdg. (Visited on 02/03/2016).

FAQ / BEM. Block, Element, Modifier / BEM. https://en.bem.info/faq. (Visited on 02/15/2016). Yandex.

Gallagher, Nicolas. *About normalize.css – Nicolas Gallagher*. http://nicolasgallagher.com/about-normalize-css/. (Visited on 02/14/2016). 2.

Hunter, Cameron et al. *SUIT CSS: style tools for UI components*. https://suitcss.github.io/. (Visited on 02/09/2016). suitcss.

Koblentz, Thierry. *Challenging CSS Best Practices – Smashing Magazine*. https://www.smashingmagazine.com/2013/10/challenging-css-best-practices-atomic-approach/. (Visited on 02/10/2016). Yahoo.

Lafont, Yves. “The linear abstract machine”. In: *Theoretical computer science* 59.1-2 (1988), pp. 157–180.

Lie, Håkon Wium. *CSS*. Mar. 2005.

Medeiros, Miller H. Borges. *SOLID CSS*. http://blog.millermedeiros.com/solid-css/. (Visited on 02/05/2016).

Media Queries. https://www.w3.org/TR/css3-mediaqueries/. (Visited on 02/03/2016). W3C.

Meyer, Eric A. *Containing Floats (Complex Spiral Consulting)*. http://www.complexspiral.com/publications/containing-floats/. (Visited on 02/14/2016). Aug. 2003.

– *CSS Tools: Reset CSS*. http://meyerweb.com/eric/tools/css/reset/. (Visited on 02/14/2016).

– *Uncollapsing Margins (Complex Spiral Consulting)*. http://www.complexspiral.com/publications/uncollapsing-margins/. (Visited on 02/14/2016). Nov. 2004.

Parasuraman, Raja and Victor Riley. “Humans and automation: Use, misuse, disuse, abuse”. In: *Human Factors: The Journal of the Human Factors and Ergonomics Society* 39.2 (1997), pp. 230–253.

Selectors Level 3. https://www.w3.org/TR/css3-selectors/#specificity. (Visited on 02/14/2016). W3C.

Sounders, Steve. *Performance Impact of CSS Selectors | High Performance Web Sites*. http://www.stevesouders.com/blog/2009/03/10/performance-impact-of-css-selectors/. (Visited on 01/23/2016).

style – style (presentation) information - HTML5. http://w3c.github.io/html-reference/style.html. (Visited on 01/24/2016). W3C.

The differences between BEM and SUIT CSS naming conventions - Stack Overflow. http://stackoverflow.com/questions/30892764/the-differences-between-bem-and-suit-css-naming-conventions. (Visited on 02/05/2016).

Thomas, David and Andy Hunt. *The Pragmatic Programmer: From Journeyman to Master*. 1999.

Use class with semantics in mind - Quality Web Tips. https://www.w3.org/QA/Tips/goodclassnames. (Visited on 02/10/2016). W3C.

Using CSS variables - CSS | MDN. https://developer.mozilla.org/en-US/docs/Web/CSS/Using_CSS_variables#Browser_compatibility. (Visited on 02/09/2016). Mozilla Foundation.

Wirth, Jan. “Analyse und Vergleich der CSS Präprozessoren LESS, Sass und Stylus im Hinblick auf benötigte Einarbeitungszeit, Praxisnutzen und Vor- bzw. Nachteile gegenüber reinem CSS”. DHBW Mosbach, Sept. 2014.

Wulf, William and Mary Shaw. “Global variable considered harmful”. In: *ACM Sigplan notices* 8.2 (1973), pp. 28–34.

Declaration of authenticity

I, Jan Wirth, certify that:

1. the paper being submitted for examination is my own account of my own research
2. where I have drawn on the work, ideas and results of others this has been appropriately acknowledged in the thesis
3. the paper being submitted for examination has not been submitted in the context of another exam

_____

Jan Wirth
Mosbach, the March 29, 2016