Martti Leino

Coding Standards in Web Development

Metropolia University of Applied Sciences

Bachelor of Engineering

Media Technology

Bachelor’s Thesis

10 April 2016

Tekijä Otsikko Sivumäärä Aika	Martti Leino Ohjelmointistandardit verkkokehityksessä 21 sivua + 1 liite 10.4.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Mediatekniikka
Suuntautumisvaihtoehto	Digitaalinen media
Ohjaajat	Lehtori Ilkka Kylmäniemi

Insinöörityön tarkoituksena oli koota ja analysoida tietoa ohjelmointistandardeista. Tutkimuksen tavoitteena oli luoda tietokooste, jonka pohjalta olisi mahdollista selvittää ja perustella ohjelmointistandardien hyötyjä ja tarpeellisuutta sekä luoda perusta ohjelmointistandardien luomiselle ja ylläpitämiselle.

Tutkimuksessa ilmeni, että ohjelmointistandardit ja erityisesti niihin sisältyvät tyylilliset seikat vaikuttavat koodin luettavuuteen ja sitä kautta ylläpidettävyyteen ja kehitettävyyteen. Ohjelmointistandardeja pidetään aikaa vievinä ja tarpeettomina, vaikka ne tosiasiassa säästävät aikaa pitkällä tähtäimellä vähentämällä koodin vaatimaa ylläpitoaikaa.

Hyvän ohjelmointistandardin seuraaminen hyödyttää kaikkia, mutta ohjelmoijat ovat usein tapoihinsa juurtuneita, mikä tarkoittaa että standardista on helppo lipsua vastahakoisuuden, tottumuksen tai aikarajoitteiden vuoksi. Tämän vuoksi tarvitaan toimenpiteitä, jotka varmistavat, että ohjelmointistandardia noudatetaan.

Insinöörityön suurimmaksi ongelmaksi osoittautui teoriatiedon hajanaisuus. Ohjelmointistandardeja koskevaa tieteellistä tutkimusta on hyvin vähän, joten siinä mielessä työ saavutti tarkoituksensa: insinöörityön tulos on tietokokonaisuus, jonka pohjalta on mahdollista hallinnoida ohjelmointistandardeja.

Avainsanat	ohjelmointistandardit, koodin laatu

Author Title Number of Pages Date	Martti Leino Coding standards in Web development 21 pages + 1 appendix 10 April 2016
Degree	Bachelor of Engineering
Degree Programme	Media Technology
Specialisation option	Digital Media
Instructor	Ilkka Kylmäniemi, Lecturer

The purpose of this final year project was to collate and analyze knowledge on the theory of coding standards. The goal of the study was to create a hub of centralized knowledge, based on which it would be possible to investigate and debate the benefits and necessity of coding standards, and to create a basis for creating and maintaining coding standards.

The study found that coding standards and especially their stylistic components impact code legibility, and consequently maintainability and extendibility. Coding standards are reproached for being time-consuming and unnecessary, while in fact they save time in the long term by reducing the time required for code maintenance.

Conforming to a good coding standard is beneficial to all, but programmers are creatures of habit, which means that they are prone to slip from the standard due to aversion, habit, or time constraints. This is why coding standards require enforcement to ensure conformity.

The greatest obstacle for this project was the sparsity of theoretic knowledge. Scientific research regarding coding standards is scarce, so in that sense the project fulfilled its purpose – in conclusion, the thesis offers a collection of information, based on which it is possible to manage coding standards.

Keywords	coding standards, code quality

Contents

1 Introduction 1

2 Coding standard 2

2.1 Enforcement 2
2.2 Automatic enforcement 3
2.3 Code reviews 3
2.4 Significance of coding standards 3

3 Coding standard conventions 6

3.1 Structural formatting 6
3.2 Braces 7
3.3 Whitespace 8
3.4 Naming conventions 9
3.5 Identifier styles 12
3.6 CamelCase vs. snake_case 14
3.7 BEM 17
3.8 Language subsetting 20
3.9 Guidelines 20

4 Conclusion 22

References 23

Appendices

Appendix 1. How to create and maintain coding standards

1 Introduction

Programming has traditionally been a job done by an individual expert. History remembers programming legends such as Linus Torvalds and Steve Wozniak. With programming becoming more mainstream and open source climbing in popularity, the software industry has moved away from the hands of solo-working code gurus into development teams, where multiple programmers work on a single project or contribute to a single codebase. [1; 2.]

Despite this change, writing code is still a very personal effort, making it an odd fit for teamwork. A team relies on communication, so it logically follows that code must be communicative in the sense that it is readily understandable for all members of the team, including the original author of the code in the future. [3.]

Coding standards have been presented as a solution to make code more communicative regardless of the prior involvement of the person reading it. This solution is however, often met with disinterest or even disdain, possibly due to its perceived impact on innovation, even though not having to spend time deciphering foreign looking code would logically seem appealing. Another explanation for aversion is that knowledge of the subject is uncommon, possibly because programming is such an organic concept and the theoretical knowledge pertaining to coding standards is scattered. [2; 3.]

Focusing on the scope of web development, the purpose of the research that this thesis documents is to collate and analyse coding standard theory in order to find out what benefit coding standards provide and whether they are necessary, and to assemble a theoretical basis for composing and maintaining coding standards.

2 Coding standard

The term coding standard refers to a set of rules related to writing code that have been standardized in some context, but are not enforced by interpreters or compilers. A coding standard defines and seeks to enforce conventions such as programming language subsets, naming conventions, and coding style for the purpose of improving code quality, which according to Paul Burden consists of the following: [4; 5.]

- safety
- security
- maintainability
- portability
- testability.

As compilers are only required to monitor the code for syntax and constraint errors, the purpose of a coding standard is to prevent errors and unwanted behaviour in the code caused by the programmer writing it. [4; 5.]

2.1 Enforcement

Coding standards are met with some resistance, because they can be seen as causing extra work, and compliance might force programmers to adopt a style to which they are not accustomed. Programmers are creatures of habit, which entails that any negatively perceived change requires a convincing rationale or a compulsion. Coding standards do not consistently provide a rationale, but rather tend to consist of concise pragmatic rules. [2; 4; 6; 7.]

A coding standard that is not enforced often goes unused. Even if programmers understand and agree with a standard, cutting corners can be tempting or deemed necessary due to time constraints or other inevitabilities. Rigid enforcement effectively prevents violations of coding standards. Enforcement can be implemented via automatic enforcement and code reviews. [2; 4; 7.]

2.2 Automatic enforcement

Automatic enforcement refers to using code analysis tools known as linters that check the code against a set of rules. A linter can be configured to use a custom coding standard and flag violations as errors. While infallible coding standard linter rules are difficult to create and unexpected loopholes might cause violations to go through unnoticed, automatic enforcement is still recommended, as it can provide fast and interactive quality control while being non-personal and thus unoffensive. [4; 6; 8.]

2.3 Code reviews

Code review refers to peer reviewing code in order to provide quality control. Code reviews are the main cause of aversion to implementing coding standards, as they are time-consuming and potentially confrontational, but the benefits of code reviews are considerable. Code reviews promote sharing of knowledge and actually save resources by weeding out bugs and problematic implementations such as an unmaintainable logical structure that would go unnoticed by a linter. The use of code reviews is advisable, though care must be taken to keep the reviews neutral and non-confrontational. Using a linter in addition to code reviews is helpful, as it provides a pre-screening and saves time on code reviews. [4; 6; 7; 9.]

2.4 Significance of coding standards

While a coding standard helps create more secure and reliable programs, the most significant improvement a coding standard offers is readability. Maintaining a piece of code constitutes 60 % of its lifetime cost on average, which means that the code will be read by a programmer numerous times after it is first written. Thus the easier the code is to read, the cheaper it ends up being. [4; 10; 11, p. 96.]

A study conducted by William Chase and Herbert Simon in 1973 found that when chess players were shown a legal position on a chess board for a short period of time, the beginners could recall the location of four pieces on average, while the experts could recall the location of all the pieces. The study was controlled for inherently superior memory by repeating the experiment with randomly placed pieces, which displayed beginner level recall performance in all the players, implying that standardized patterns significantly increase human ability to process visual information. Thus in order to maximize code maintainability, the code should be consistent in style and structure, and conform to recognizable patterns. [10; 12.]

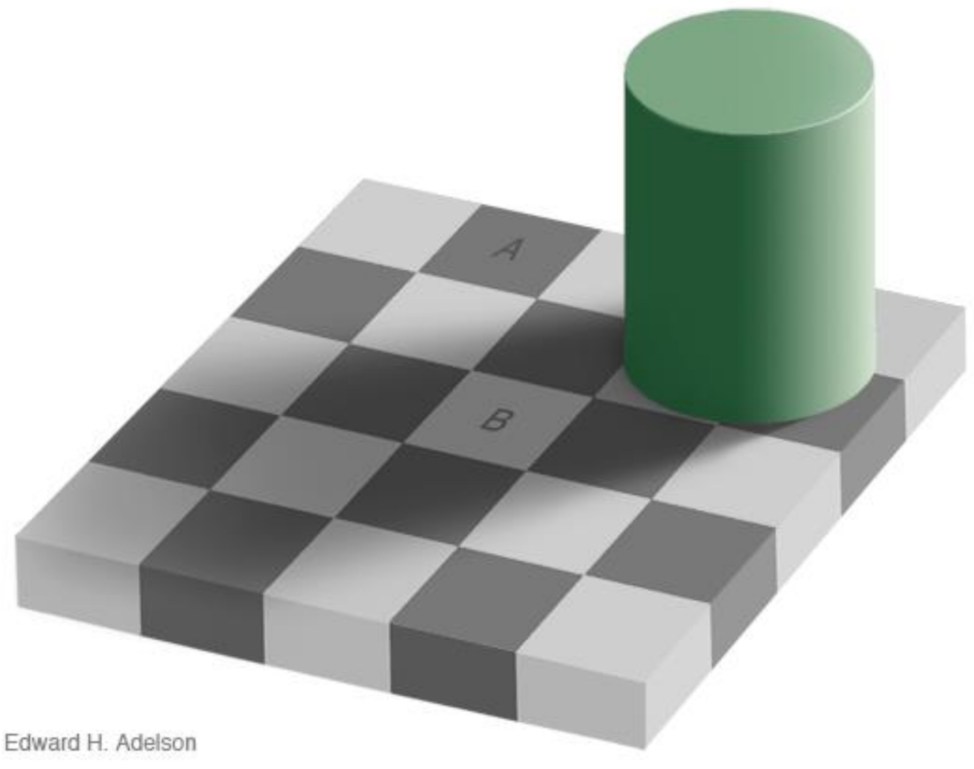


Figure 1. The checkershadow illusion [13].

A psychologist by the name of Daniel Kahneman came up with a model of two systems of thinking. System 1 is associative and fast, while system 2 is analytic and slow. Optical illusions such as the checkershadow illusion seen in figure 1 are caused by the interaction of these two systems. System 2 takes time to analyze data, so quick assessment is done by system 1 by replacing a demanding problem such as processing an image with a simpler problem and solving that, then passing the data on to system 2. System 2 could find that the A and B squares in the illusion are the same color through analysis, but system 2 operates on the approximate assessment made by system 1, which in this case is the false assumption that the squares are a different color. This causes system 2 to search for a rationale for the data it received and when it finds a logical explanation, which is the shadow, it decides that the assessment is correct. Applied to coding, this theory suggests that visually consistent and meaningfully formatting makes assessing code quicker and more precise, due to system 1 being able to provide system 2 with accurate data. [2; 3; 6.]

3 Coding standard conventions

3.1 Structural formatting

Structural formatting of a code document refers to formatting the code structure in a meaningful way, encompassing subjects such as indentation of block statements, syntactically insignificant whitespace, and brace placement. Code formatting generally holds no meaning to code interpreters – the functions in listing 1 perform identically, but the omission of structural formatting renders the function virtually illegible. [2; 14.]

```
function get_string_file_size($string, $encoding) {
    $byte_count = mb_strlen($string, $encoding);

    if($byte_count >= 1024) {
        $file_size = round($byte_count / 1024, 2).'
KB';
    }
    else {
        $file_size = $byte_count.' bytes';
    }

    return $file_size;
}

// No structural formatting
function
get_string_file_size($string,$encoding){$byte_count=mb
_strlen($string,$encoding);if($byte_count>=1024){$file
_size=round($byte_count/1024,2).'
KB';}else{$file_size=$byte_count.'
bytes';}return$file_size;}
```

Listing 1. Example PHP function with and without structural formatting.

Structural formatting is essentially a matter of programmer preference, considering that it has no direct parallel in natural written language and code interpreters are indifferent to syntactically insignificant formatting. Structural formatting is however, an integral factor in code comprehension, as it provides visual cues for recognizing program components and their internal logic. Programming is based on the learning and application of models, which in combination with the necessity of structural formatting implies that while there is no right or wrong formatting style, a formatting style should be defined and used consistently. [2; 15; 16.]

3.2 Braces

The use and positioning of braces in function declarations and control flow statements has been a topic of debate since they were introduced in programming languages. The debate consists of two points: whether optional braces can be omitted and whether an opening brace should be left or right, as demonstrated in listing 2. [6; 17.]

```
// Opening brace to the right
function foo() {
    return {
        bar: true
    };
}

// Opening brace to the left
function foo()
{
    return
    {
        bar: true
    };
}
```

Listing 2. JavaScript function declarations exemplifying the left and right brace conventions.

Both conventions of opening brace position are used in all web development languages and as interpreters are not affected by this decision, it is a question of preference. JavaScript however, provides an exception due to an oversight in language design – JavaScript implements a feature called ASI (Automatic Semi-colon Insertion). ASI adds a semi-colon to end a statement when it encounters an unallowed token after said statement. The first function in listing 2 returns an object literal where bar equals true, as expected. In the second function ASI inserts a semicolon after the return statement, and the function silently fails, returning undefined. As it fails without producing an error, debugging it is challenging and possibly very time consuming. As the brace convention should be consistent throughout the code, this JavaScript error is the single reason why it is advisable to use the right side brace convention. [6; 16; 17; 18; 19.]

```
if($foo == $bar){
    echo $first;
}

if($foo == $bar)
    echo $first;

// The second statement
    echo $second;
```

Listing 3. Single-statement control structures in PHP with and without braces.

Optional braces refer to single-statement control structures, where braces are not required by the interpreter, illustrated in listing 3. Both if-blocks work the same, but if a second statement is added after the first echo-statement, it will only be executed conditionally by the first block. The second if-block will execute the added second statement regardless of whether the condition in the if-statement is met, thus breaking the control flow logic. Avoiding omitting braces is generally met with some resistance, but program logic requires precision and the human brain is not infallible, which is why it is considered good practice to always include braces in control structures to reduce obscurity. [2; 6; 20.]

3.3 Whitespace

Whitespace in programming refers to syntactically insignificant blank characters in code, such as the space, tab, and line break characters, making whitespace the cornerstone of structural formatting. Whitespace can be used to format code for readability with conventions such as adding spaces to comma separated lists, adding line breaks between semantically separate code blocks, and adding spaces between tokens in a list of conditions or an operation statement. Established coding standards differ in whitespace rules – for example Drupal coding standards allow no space between control structure parenthesis and their inner statements, while WordPress coding standards require a space before and after each parenthesis. Even though there are no clear winners among whitespace conventions, it is considered good practice to adhere to the whitespace conventions already utilized in the code when extending existing code or adding to a codebase. [2; 6; 14; 18; 21; 22.]

While other whitespace conventions are largely matters of preference, indentation of code is considered a de facto standard in structural formatting among programming languages that support it. Indentation emphasizes the logical structure of the code and is seen as an integral part of code readability. The prevalent question within indentation is whether to use tabs or spaces. The WordPress coding standard states that tabs should always be used for indentation, while the Drupal coding standard calls for two space indentation and no tabs. The global programmer community is equally in disagreement on the matter, but there are no unrefuted arguments for either convention, so it is a matter of preference. The conventions should however, not be mixed within a single code document, since tab-stops are not standardized. This means that if the code is written on an editor where tabs are four spaces wide and then viewed on an editor where tabs are eight spaces wide, it can obfuscate the logic structure and reduce readability. [2; 18; 21; 22; 23, p. 37.]

3.4 Naming conventions

Identifier naming is perhaps the most noticeable part of a programmers touch on a piece of code, as it is left completely to the discretion of the person writing the code. Identifiers are used for multiple functional parts of a program:

- Variables
- Methods
- Classes
- Constants

An identifier has a twofold role. On one hand it provides a unique handle for the code interpreter, be it a compiler or a web browser. On the other hand it provides a canvas for passing on information about the function of the identified program element. For example in the Java coding standards by Oracle, the formatting of the name implies the type of the program element – a class name is written in mixed case with the first letter of each word capitalized, while a constant name is written in all uppercase with underscores between the words, as seen in listing 4.

```
class SledgeHammer;

static final int MAX_WEIGHT = 9001;
```

Listing 4. Example declarations of a class and a constant in Java.

An identifier should indicate the purpose of a program element to the reader of the code. Using whole words instead of acronyms and keeping identifiers descriptive, meaningful and mnemonic will help keep the code easy to read, as illustrated in listing 5. [3; 24.]

```
// Classes
class ColorPicker; // Simple and descriptive.
class cp; // Conveys no meaningful information and is
not identifiable as a class.
class MyClass; // Does not identify its purpose.

// Methods or functions
drawColorPickerPalette(); // Clearly indicates its
purpose.
randomWord(); // Not recognizable as a function, could
as well be a variable that has been assigned a random
word.
```

Listing 5. Examples of identifiers in Java.

Naming convention should be thought of as a means of communication between programmers reading the code, as it impacts the readability of the code and thus the rate at which a programmer can effectively work on it. Standardizing a naming convention that enforces semantic, descriptive, and readable identifiers will improve coding efficiency, since an observer reading the code will instantaneously be aware of the purpose of program elements as they come along, instead of having to spend time cross-referencing variables and functions across the code. [3.]

> Programs must be written for people to read, and only incidentally for machines to execute [23, p. 12].

A programming language is not a computer language, but a human-readable medium of logical expression, that is used to merely communicate instructions to a computer. Identifier length has no impact on performance in compiled languages and only negligible impact in interpreted scripting languages like PHP or JavaScript, so abbreviating or otherwise shortening identifiers in development code serves no real purpose, while doing real harm to coding efficiency. As an extreme example, the two versions of the example function illustrated in listing 6 are equal in terms of program execution, but replacing the identifiers with single characters adversely affects human readability. [23; 25; 26.]

```
var calculateAverageStringLength = function() {
    var totalStringLength = 0;
    var averageStringLength;
    if(arguments.length > 0) {
        for(var i = 0; i < arguments.length; i++) {
            totalStringLength += arguments[i].length;
        }
        averageStringLength = totalStringLength /
arguments.length;
    } else {
        averageStringLength = 0;
    }
    return averageStringLength;
}

// Single character identifiers
var calculateAverageStringLength = function(){
    var n = 0;
    var e;
    if(arguments.length > 0) {
        for(var i = 0; i < arguments.length; i++) {
            n += arguments[t].length;
        }
        e = n / arguments.length;
    } else {
        e = 0;
    }
    return e;
};
```

Listing 6. Example JavaScript function with expressive identifiers and single character identifiers.

3.5 Identifier styles

The need for multi-word identifiers becomes apparent when considering identifier descriptiveness. Identifiers must be unique within their scope, so when using single-word identifiers to name for example PHP arrays that serve a similar purpose while preserving descriptiviness, the options boil down to either a limited pool of synonyms or non-semantic arbitrary identifier variation. Arbitrary identifier variation would require comments or other means of documentation for distinction, as illustrated in listing 7. The larger the codebase of a program grows, the less expressive single-word identifiers get, which in turn hinders coding efficiency due to the need to cross-reference the documentation on identifiers to find out their detailed purpose, as opposed to getting all the necessary information from the identifier. [14.]

```php
// Native languages in Europe
$languages1 = array();

// Native languages in Asia
$languages2 = array();

// Native languages in Africa
$languages3 = array();



$native_languages_europe = array();
$native_languages_asia = array();
$native_languages_africa = array();
```

Listing 7. Example of array declarations in PHP.

Most programming languages – including all of the common languages used in web development – are whitespace delimited. This means that these programming languages recognize identifiers as individual tokens when they are delimited by whitespace or reserved characters denoting literals, separators, operators or escape sequences. In languages where whitespace is considered a delimiter, identifiers cannot contain whitespace characters. The first function declaration in listing 8 causes the JavaScript interpreter to throw a syntax error due to an unexpected identifier. The second declaration is syntactically correct, but presents a problem with legibility. [14; 27, p. 83–100.]

```
function align editor in opera() { ... }

function aligneditorinopera() { ... }
```

Listing 8. Example of syntactically erroneous and correct function declarations in JavaScript.

The purpose of identifier style formatting is to solve the problem of illegible multi-word identifiers by specifying a method of delimiting separate words within an identifier. The two most prevalent identifier styles in programming to date are camelCase and snake_case, due to being popular conventions in the most commonly used programming languages, including Java, Python, PHP, the C family and JavaScript. The prevalence of these two naming conventions holds especially true for web development, considering native JavaScript and Java methods use camelCase and native PHP functions use snake_case, prompting the web developer community to heavily favor the aforementioned naming conventions for the sake of homogeneity among identifiers within a code document. [14; 24; 28; 29; 30.]

Most programming languages do not allow hyphens in identifiers, since the hyphen is generally tokenized as the subtraction operator. A notable exception in web development is CSS. CSS properties use the hyphen-delimited lisp-case, named after the convention used in Lisp; a programming language dating back to 1958. CSS originally only allowed underscores in identifiers if they were escaped, which was poorly supported by browsers at the time, so programmers shifted away from underscore delimited identifiers. Modern CSS allows underscores in identifiers, but naming conventions in HTML are still a factor, since CSS only provides presentation rules for HTML elements. Naming convention in HTML is affected by the fact that hostnames in URLs may not contain underscores, and the use of underscores in URL structures is discouraged as it affects search engine optimization. [31; 32; 33; 34.]

3.6 CamelCase vs. snake_case

One of the most commonly debated aspects of coding conventions in web development is the question of superiority between camelCase and snake_case. CamelCase or medial capitals is an identifier style where separate words within an identifier are in lowercase, but are delimited by capitalizing the first letter of each internal word. The first word is also commonly capitalized when naming classes in object-oriented programming. Snake_case refers to an identifier style where individual words are in lowercase, delimited by underscores. [24; 30; 35.]

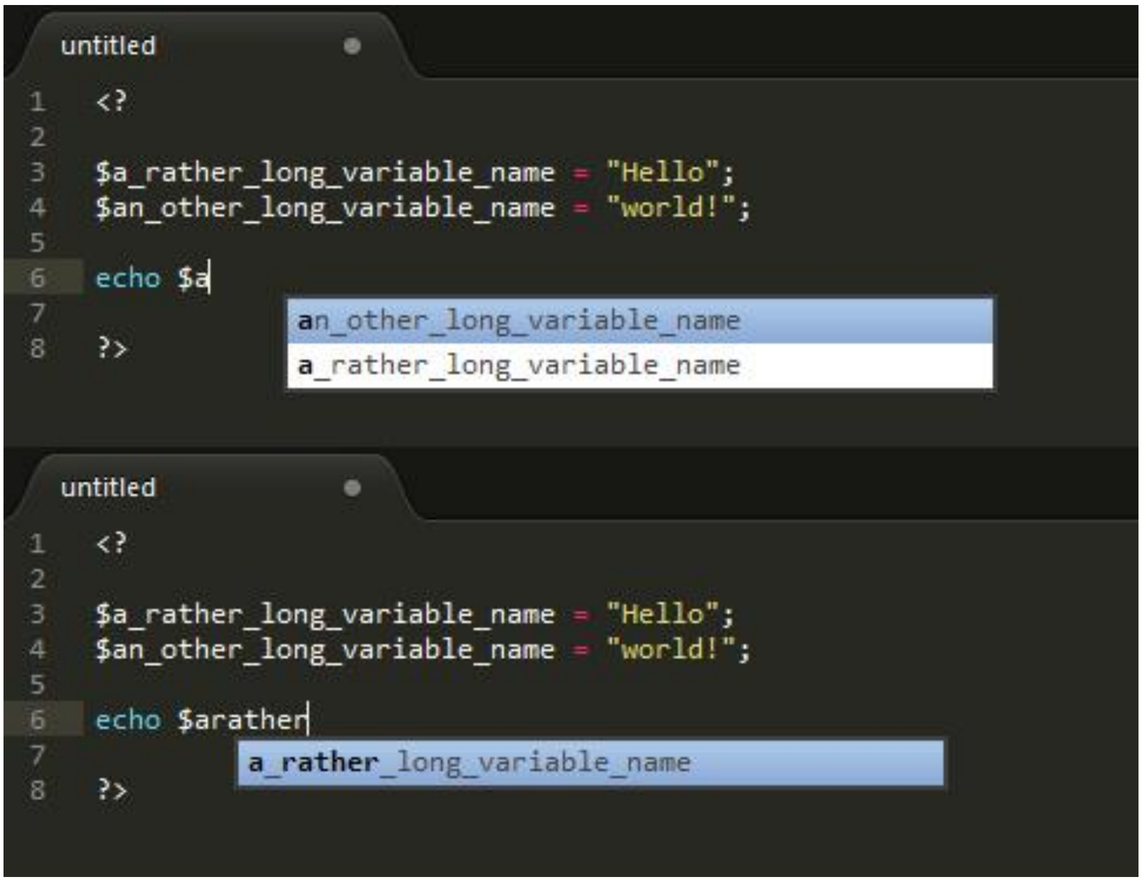
```
function getNumberOfSkinCareEligibleItems() { ... }

function get_number_of_skin_care_eligible_items() { ... }

class Foo {
    function aMemberFunction($foo, $bar, $baz) {
        for($i = 0; $i < count($foo); $i++) {
            if($foo[$i] == $bar) {
                for($j = 0; $j < count($baz); $j++) {
                    if($camelCase == true) {
                        getNumberOfSkinCareEligibleItems();
                    }
                    else if($snake_case == true) {
                        get_number_of_skin_care_eligible_it
                        ems();
                    }
                }
            }
        }
    }
}

function isIllicitIgloo() { ... }

function is_illicit_igloo() { ... }
```

Listing 9. Extreme examples of differences between camelCase and snake_case.

Listing 9 demonstrates some extreme examples commonly referenced when comparing camelCase and snake-case. Long identifiers are often necessary for descriptiveness, but can be slow to process at a glance. Snake_case makes the identifier longer due to extra characters. Especially when deeply nested, this makes the identifier more likely to either go off screen or wrap to a new line, depending on the source code editor settings. Underscores in identifiers however, closely mimic linguistically natural whitespace, which reduces their impact on reading time. [35; 36.]

The main argument against camelCase addresses the issues with readability. Discerning individual words in camelCase identifiers becomes significantly harder when the identifier contains graphically similar characters, such as seen in the last two function declarations in listing 9. Short words and standardized uppercase acronyms, such as TCP and IP, present another issue with camelCase readability. Retaining capitalization in an identifier like TCPIPSocket impacts readability, while conforming to the camelCase convention with an identifier like tcpIpSocket makes it more difficult to identify the words as acronyms. In contrast to camelCase, snake_case does not suffer from reduced readability when containing uppercase acronyms. [35; 36.]

CamelCase has been shown to be 20 % slower to read than snake_case. Research also indicates that camelCase may provide a larger probability of correctness and getting used to reading identifiers in one style has a negative impact on reading other styles. Correctness in reading and writing identifiers has however, become virtually a non-issue due to modern source code editors implementing auto-completion features, as seen in figure 2. While attunement to one identifier style does decrease speed when reading other styles, it is inconsequential in web development, considering that full-fledged web development calls for proficiency in all of the common languages used in web development covering both camelCase and snake_case in their respective native identifier styles. [35; 36; 37; 38.]



```
<?

$a_rather_long_variable_name = "Hello";
$an_other_long_variable_name = "world!";

echo $a

?>
```

an_other_long_variable_name
a_rather_long_variable_name
untitled

```
<?

$a_rather_long_variable_name = "Hello";
$an_other_long_variable_name = "world!";

echo $arather

?>
```

a_rather_long_variable_name

Figure 2. Auto-completion of identifiers in Sublime Text 2.

While camelCase is widely considered more aesthetically pleasing, snake_case is proven to be slightly more efficient in modern development environments. Eye tracking research has also shown that reading camelCase requires more visual effort, as subjects fixate on camelCase identifiers for a longer duration. Contrary to what research indicates, it is generally considered advisable to follow the native convention of the programming language being used. [35; 36; 37; 38.]

3.7 BEM

BEM is a methodology that was created by the russian search engine company Yandex. BEM introduces a naming convention, which primarily aims to reduce CSS codebase and make collaboration on the code more efficient by enforcing meaningful, descriptive, and functional CSS selectors. The modularity inherent to the methodology allows for creation of web-component libraries for fast and efficient web development. The term BEM is an acronym of the methodology’s key elements. [39; 40.]

Block

Blocks are the basic elements of the BEM methodology. A block identifies a baseline entity that holds content or other entities and is inherently meaningful. [39; 40.]

Element

An element is a context dependent child of a block. The element is not meaningful or descriptive on its own, but specifies an entity as a functional part of the block. [39; 40.]

Modifier

A modifier denotes an alternate version of a block or an element. Modifiers are used to change the appearance or behavior of the entity they are attached to. [39; 40.]

The cornerstone of the BEM methodology is reusability. BEM aims to create components that are modular and infinitely reusable, by dividing an interface into logically independent blocks. Thus the notation uses HTML class-attributes as identifiers instead of the unique id-attributes. BEM identifiers conform to the format block__element--modifier, where the element is separated from the block by two underscores and a modifier can be appended to either part, separated by two hyphens. There are multiple styles for the separators, but block__element--modifier is the most widespread due to being easily identifiable as BEM separators, while still allowing for hyphens or underscores to be used as delimiters in multi-word identifiers for entities. [39; 40; 41; 42.]

Modularity is considered a requirement for maintainable systems. The methodology achieves component modularity by reducing CSS selector specificity. Deeply nested CSS selectors are discouraged in BEM, as they are susceptible to errors, difficult to maintain, difficult to interpret at a glance, and overspecific, making them cumbersome to override. Minimally specific selectors that are agnostic to HTML tags and use class names that are independent of content achieve their purpose regardless of position in the Document Object Model, which is an interactive representation of the web document. [39; 40; 41; 42.]

```
<ul class="list list--bullet">
    <li class="list__item"></li>
    <li class="list__item"></li>
    <li class="list__item"></li>
</ul>

.list {
    list-style: none;
    font-size: 12px;
    border: 1px solid black;
}

.list__item {
    margin-bottom: 10px;
}

.list--bullet {
    list-style: disc;
    font-style: italic;
}
```

Listing 10. HTML and CSS snippets utilizing the BEM methodology.

As seen in listing 10, it is not necessary to see the HTML document to understand the purpose of the BEM CSS selectors. This exemplifies the ideology behind BEM: a piece of CSS markup should tell the programmer what it does by its identifier alone, without the need to reference the HTML document in which it is used. A programmer who recognizes the BEM naming convention can instantly tell that the list__item -class stylizes an item inside the list-block. As opposed to using a class named bullet for the bulleted list, which is a common implementation style, the list--bullet -selector clarifies to the reader that it is used to modify the style of an HTML element with the list-class. The BEM methodology promotes fluid and modular design and improves CSS readability and maintainability. [39; 40; 41; 42.]

3.8 Language subsetting

Language subsetting in coding standards means creating a restricted set of features that are allowed to be used when coding in said language. Designing a programming language is not a simple or infallible process, and languages are often published with design flaws or faulty implementations. Once a language is published and adopted by programmers, the designer can not go back and change defective functionalities, but programmers can subset the language. [6.]

> Only a madman would use all of C++ [6].

All programming languages contain some amount of oversights. For example the extract-function in PHP is a powerful tool used to import items in an associative array as independent variables into the current symbol table, but it creates a vulnerability if used on untrusted data – like the superglobal $_GET variable. Extract also lacks transparency, meaning that it is difficult to discern what variables are imported by the function when encountering it in a code document. Creating a subset of a programming language is a labor intensive process and requires intimate knowledge of the language, so researching existing coding standards is an advisable alternative to reinventing the wheel. [6; 30; 43.]

3.9 Guidelines

Creating good coding standards is a demanding process. The objective of a coding standard is to improve code quality, so a question that should run alongside the process is as follows:

> Does this rule actually help? [4.]

It is easy to overregulate coding standards and add rules without inherent value simply for the sake of adding rules. Rules should not be added based on personal opinion as they are indefensible. Instead each individual rule should have an explained rationale behind it to reduce programmer aversion to the coding standard and reduce confusion and room for interpretation. A rule should demonstrably improve code quality to merit its inclusion. As aversion is unlikely to be extinguished, a coding standard should include means of enforcing it. One argument against coding standards is that they are dangerous, as they can introduce bugs in legacy code, which suggests that great care should be exercised if new coding standards are applied to legacy code, and refactoring should not be done without a compelling reason. [4; 6; 21; 44.]

When creating a coding standard and especially when subsetting a programming language, it is advisable to refer to existing coding standards. Creating new niche standards is counterintuitive, as a lot of coding reference comes from the online developer community, and a standard used by a handful of programmers is less standard than one used globally. A coding standard that is unreasonably far from more common coding standards will also have an impact on programmers who are introduced to the standard in the future, making adaptation more difficult. As naming conventions and the structural formatting of the code have a significant effect on code maintainability, legibility and development speed, they should be implemented conscientiously. [2; 3; 4; 6; 44.]

4 Conclusion

The purpose of this thesis was to compile theoretical knowledge on coding standards, to assemble a theoretical basis for composing and maintaining them, and to answer to questions:

- What benefit coding standards provide?
- Are coding standards necessary?

The research indicated that coding standards not only make code more resilient to errors and more secure, but when properly applied and enforced, they greatly improve readability, maintainability and development speed. Coding standards also promote learning due to the need to understand the coding standard and due to the shared knowledge created by code reviews.

Whether coding standards are necessary is debatable. A solo developer might not see any value in adhering to a coding standard, and nothing in programming explicitly requires coding standards, but this does not imply that coding standards aren't valuable. Even though enforcing coding standards is thought to be time-consuming and programmers tend to rebel when presented with coding standards that are not in line with their personal preferences, following a coding standard greatly cuts down on time spent on fixing, refactoring and tuning code, which in turn reduces costs and increases productivity. Furthermore, code written under a good coding standard will be understandable even after nobody remembers what it does.

The theoretical basis for composing and maintaining coding standards proved to be more of a challenge than initially assumed. Knowledge of coding standard theory is indeed scattered in tiny morsels around the collective knowledge of the web developer community, and while there are numerous well-established coding standards for different programming languages and libraries, they are not tremendously explicative, and judging by the lack of actual specialized theory on coding standards, composing coding standards seems to be an organic process of trial and error. The basis however, does provide an operable frame of thought to guide the process of creating or refining coding standards.

References

1 StackExchange. Are there any famous one-man-army programmers [online]. URL: http://programmers.stackexchange.com/questions/47197/are-there-any-famous-one-man-army-programmers. Accessed 10 April 2016.

2 Leino, Jukka. Head Consultant, JAL-Ware open company, Espoo. Conversation. 2 April 2016.

3 Zakas, Nicholas. Why Coding Style Matters [online]. October 2008. URL: https://www.smashingmagazine.com/2012/10/why-coding-style-matters/. Accessed 9 April 2016.

4 Burden, Paul. Coding Standard Compliance – Some Facts and Some Fallacies [online]. March 2012. URL: http://www.programmingresearch.com/resources/seminars/coding-standard-compliance-some-facts-and-some-fallacies/. Accessed 10 April 2016

5 MSDN. Coding Standards and Code Reviews [online]. URL: https://msdn.microsoft.com/en-us/library/aa291591(v=vs.71).aspx. Accessed 10 April 2016.

6 Crockford, Douglas. Programming Style & Your Brain [online]. 2012. URL: https://www.youtube.com/watch?v=_EANG8ZZbRs. Accessed 7 April 2016.

7 Barr, Michael. How to enforce coding standards automatically [online]. July 2011. URL: http://www.embedded.com/electronics-blogs/barr-code/4218283/How-to-enforce-coding-standards-automatically. Accessed 10 April 2016.

8 Grotewold, Blake. About SublimeLinter [online]. 2015. URL: http://www.sublimelinter.com/en/latest/about.html. Accessed 10 April 2016.

9 Huston, Tom. What is code review [online]. URL: https://smartbear.com/learn/code-review/what-is-code-review/. Accessed 9 April 2016.

10 Dorman, Scott. Why Coding Standards Are Important [online]. URL: http://scottdorman.github.io/2007/06/29/Why-Coding-Standards-Are-Important/. Accessed 1 April 2016.

11 Glass, Robert. Facts and Fallacies of Software Engineering. Boston. Addison-Wesley Professional, 2002.

12 Chase, William & Simon, Herbert. The Mind’s Eye in Chess. Proceedings of the Eighth Annual Carnegie Symposium on Cognition 1972. 215–281.

13 Adelson, Edward. The checkershadow illusion [online]. 1995.
URL: http://web.mit.edu/persci/people/adelson/checkershadow_illusion.html.
Accessed 10 April 2016.

14 Cunningham & Cunningham, Inc. Syntactically Significant Whitespace Considered Harmful [online].
URL:
http://c2.com/cgi/wiki?SyntacticallySignificantWhitespaceConsideredHarmful.
Accessed 4 April 2016.

15 Udden, J., Hammarström, H. & Jansen, R. What is the similarity between a natural languages and programming languages [online].
URL: http://www.mpi.nl/q-a/questions-and-answers/wat-zijn-de-overeenkomsten-en-verschillen-tussen-natuurlijke-talen-en-programmeertalen. Accessed 10 April 2016.

16 Valid-computing.com. Control Structures [online].
URL: http://www.valid-computing.com/control-structures.html. Accessed 10 April 2016.

17 StackExchange. Should curly braces appear on their own line [online].
URL: http://programmers.stackexchange.com/questions/2715/should-curly-braces-appear-on-their-own-line. Accessed 10 April 2016.

18 WordPress.org. PHP Coding Standards [online].
URL: https://make.wordpress.org/core/handbook/best-practices/coding-standards/php/. Accessed 2 April 2016.

19 Allardice, James. Understanding automatic semi-colon insertion in JavaScript [online]. 2012.
URL: http://jamesallardice.com/understanding-automatic-semi-colon-insertion-in-javascript/. Accessed 10 April 2016.

20 StackExchange. Single statement if block - braces or no [online].
URL: http://programmers.stackexchange.com/questions/16528/single-statement-if-block-braces-or-no. Accessed 10 April 2016.

21 StackExchange. What is the best way to use whitespace while programming [online].
URL: http://stackoverflow.com/questions/3039357/what-is-the-best-way-to-use-whitespace-while-programming. Accessed 9 April 2016.

22 Drupal Community Documentation. Coding Standards [online].
URL: https://www.drupal.org/coding-standards. Accessed 4 April 2016.

23 Abelson, Harold & Sussman, Gerald. Structure and Interpretation of Computer Programs. Cambridge. The MIT Press, 1996.

24 Oracle Corporation. Naming Conventions [online].
URL: http://www.oracle.com/technetwork/java/codeconventions-135099.html.
Accessed 7 April 2016.

25 StackExchange. Do variable names affect the performance of websites [online].
URL: http://programmers.stackexchange.com/questions/92556/do-variable-names-affect-the-performance-of-websites. Accessed 8 April 2016.

26 StackExchange. Does minified JavaScript improve performance [online].
URL: http://stackoverflow.com/questions/1181447/does-minified-javascript-improve-performance. Accessed 8 April 2016.

27 Friesen, Jeff. Java 2 by Example. 2nd ed. Indianapolis. Que Publishing, 2002.

28 Carbonnelle, Pierre. PYPL PopylaritY of Programming Language [online].
URL: http://pypl.github.io/PYPL.html. Accessed 8 April 2016.

29 W3schools. JavaScript Functions [online].
URL: http://www.w3schools.com/js/js_functions.asp. Accessed 9 April 2016.

30 The PHP Group. Php.net [online].
URL: http://php.net/. Accessed 9 April 2016.

31 Feinler, E., Harrenstien, K. & Stahl, M. DoD Internet host table specification [online]. 1985.
URL: http://tools.ietf.org/html/rfc952. Accessed 8 April 2016.

32 Google Inc. Keep a simple URL structure [online].
URL: https://support.google.com/webmasters/answer/76329. Accessed 5 April 2016.

33 StackOverflow. Why are dashes preferred for CSS selectors / HTML attributes [online].
URL: http://stackoverflow.com/questions/7560813/why-are-dashes-preferred-for-css-selectors-html-attributes. Accessed 8 April 2016.

34 Cunningham & Cunningham, Inc. Common Lisp [online].
URL: http://c2.com/cgi/wiki?CommonLisp. Accessed 4 April 2016.

35 Jeuris, Steven. CamelCase vs underscores: Scientific showdown [online]. 2011.
URL: https://whathecode.wordpress.com/2011/02/10/camelcase-vs-underscores-scientific-showdown. Accessed 3 April 2016.

36 Binkley, D., David, M., Lawrie, D. & Morrell, C. To camelcase or under_score. Proceedings of the 2009 IEEE 17th International Conference on Program Comprehension, 2009. 158–167.

37 Fekete, George. Being a Full Stack Developer [online]. September 2014.
URL: http://www.sitepoint.com/full-stack-developer/. Accessed 9 April 2016.

38 Maletic, J & Sharif, B. An Eye Tracking Study on camelCase and under_score Identifier Styles. Proceedings of the 2010 IEEE 18th International Conference on Program Comprehension, 2010.196–205.

39 Get BEM. Introduction [online].
URL: http://getbem.com/introduction/. Accessed 8 April 2016.

40 Mack, Kevin. MindBEMding – Rething Web Development [online]. August 2014.
URL: https://www.youtube.com/watch?v=vgg-NsKZaE4. Accessed 3 April 2016.

41 Hayward Mat. Modular CSS with Sass & BEM [online]. January 2014.
URL: http://mathayward.com/modular-css-with-sass-and-bem/. Accessed 9 April 2016.

42 Drupal Community Documentation. CSS architecture (for Drupal 8) [online]. 2013.
URL: https://www.drupal.org/node/1887918. Accessed 8 April 2016.

43 StackOverflow. What is so wrong with extract() [online].
URL: http://stackoverflow.com/questions/829407/what-is-so-wrong-with-extract. Accessed 2 April 2016.

44 StackExchange. Creating a coding standards document [online].
URL: http://programmers.stackexchange.com/questions/196706/creating-a-coding-standards-document. Accessed 10 April 2016.

How to create and maintain coding standards

1. Do not reinvent the wheel. Refer to existing standards:
    a. when subsetting a language
    b. to avoid creating a niche standard.
2. Document the coding standard so that each rule is strongly reasoned and resistant to interpretation.
3. Only implement rules that are helpful for improving code quality.
4. Do not implement rules for the sake of having rules – do not overregulate.
5. Do not implement rules based on personal opinions.
6. Focus the coding standard at future development; avoid applying it to legacy code.
7. Standardized coding style is your best friend. Make sure to:
    a. implement explicit structural formatting rules
    b. specify a naming convention in enough detail to produce meaningful identifiers
    c. consider enforcing the BEM methodology for front end development.
8. Introduce enforcement of the coding standard with linters and code reviews.