A SURVEY OF CASCADING STYLE SHEETS COMPLEXITY METRICS

John Gichuki Ndia[1,2], Geoffrey Muchiri Muketha[2] and Kelvin Kabeti Omieno[3]

[1]School of Computing and Informatics, Masinde Muliro University of Science and Technology, Kenya
[2]School of Computing and Information Technology, Murang'a University of Technology, Kenya
[3]School of Computing and Information Technology, Kaimosi Friends University College, Kenya

ABSTRACT

Cascading style sheets (CSS) is a Web-based style sheet language that is used for the presentation of Web documents. CSS has advanced from CSS1 to CSS3.and extensions to CSS known as CSS pre-processors have also emerged in the last few years. As is the case with regular software, CSS have inherent complexity that keeps on increasing with age which is undesirable, and metrics are needed to measure with the aim of controlling it. Although several Web metrics have been proposed in the literature, the area of stylesheets is still lagging. Findings show that few CSS-related metrics exist, and there is no evidence of proof for their mathematical soundness through the popularly known frameworks such as Briand framework and Weyuker's properties. In addition, they have not been empirically validated. In order to address this gap, future studies should focus on defining and validating new metrics for CSS and its pre-processors.

KEYWORDS

CSS, CSS Pre-processors, complexity metrics, theoretical validation, empirical validation

1. INTRODUCTION

This Cascading style sheets (CSS) is an integral part of a Web-based application and its purpose is to separate content from presentation [1]. Basically, CSS language makes it possible to style Web pages on themes such as the use of colors, fonts, and layout [2]. As is the case with regular software, CSS and its extensions have inherent complexity that keeps on increasing with age. High complexity is undesirable and leads to unreliable, difficult to maintain and understand, and error-prone CSS code [3, 4]. Therefore, several complexity metrics have been proposed to measure and consequently, control CSS code complexity [4]. The complexity in CSS has assumed causal factors such as size, rule block structures varieties, rule block reuse, cohesion and attributes definition [4].

In several studies software complexity metrics have been defined to guarantee the development of high-quality software [5]. These metrics have targeted different software domains such as procedural programming, object-oriented programming, Web services, and business processes. For example, the McCabe cyclomatic complexity metric [6] based on the control flow representation of the program has been used in both procedural and object-oriented system measurement. Chidamber and Kemerer [7] proposed a metrics suite for measuring object-oriented systems while Misra et al. [8] defined a cognitive complexity metrics suite for object-oriented systems. In the Web-based domain Pichler et al. [9], proposed complexity metrics for XML

DOI: 10.5121/ijsea.2019.10303

schema while Baski and Misra [10] have proposed complexity metrics for eXtensible Mark-up Language (XML) Web services.

Although several Web metrics have been proposed in the literature, there is a lack of good documentation on measuring CSS-related complexity. In this study, a survey of CSS structural properties, web-based metrics, and associated tool support, is conducted. The purpose of this study is to elaborate on CSS structural complexity, existing web and CSS metrics, theoretical and empirical validation of the metrics and whether these metrics have any tool support.

The rest of this paper is structured as follows. Section two presents concepts on the structural properties of CSS, Section three presents the steps of defining and validating metrics for measuring CSS-based Web complexity, Section four presents existing complexity metrics, and Section five presents conclusion and recommendations.

2. UNDERSTANDING THE STRUCTURAL PROPERTIES OF CSS

2.1. CSS and its Pre-Processors

CSS is a standard language used by front Web developers to define the look and feel of structured documents written in HTML and eXtensible Mark-up Language (XML). CSS is composed of a sequence of style rules, where each rule has a selector that selects the elements needed to style in the HTML or XML document [11].

CSS has evolved over the years from CSS1 to CSS3. More recent extensions of CSS have also been proposed such as CSS pre-processors. CSS1 was a simple version with about 50 properties and is mostly used for screen-based presentations. CSS2 includes all CSS1 properties plus an additional around 70 properties of its own. The additional properties have for instance enabled CSS2 to describe aural presentations and page breaks that couldn't be done earlier. An enhanced CSS 2.1 was also released that added more features such as the ability to describe the parts that are supported by two or more browsers [12]. Finally, CSS3 is organized into modules such as backgrounds and borders, selectors, text effects, box model, image values, and replaced content, animations, 2- and 3-dimension transformations, multiple column layouts, and user interface [11]. The purpose of modularization is to have multiple specifications, where each specification has its own progression path.

Post CSS3 developments have taken the direction of CSS pre-processors. Several CSS pre-processors have so far been proposed such as Syntactically Awesome Style Sheets (SASS), Less, Stylus, CSS-Crush, Myth and Rework with each of them having unique syntaxes [1, 13]. CSS pre-processors add extra features to those found in regular CSS such as the use of variables, nesting of rules, use of mixins, use of function calls, inheritance, use of control flow statements and use of operators [1].

Variables are defined to store one or more style values and represent data, such as numeric values and characters [1]. A variable enables reuse of the style values stored in the stylesheets. In stylesheets, variables can be used to set up colors and fonts [14]. In some instances, variable values are manipulated using arithmetic operators and by passing them to pre-processor built-in function [1]. Rule nesting is like class nesting in object-oriented programming. According to Mazinanian and Tsantalis [1], CSS Pre-processors permit a rule to be placed inside another rule as a way of combining multiple CSS rules within one another.

Some powerful features introduced by CSS pre-processors include mixins and user-defined functions. While mixins store multiple values, functions are invoked to allow the use of

parameters [1, 14]. Mixins and functions are beneficial in that they help to avoid writing repetitive codes [1]. Other new features include inheritance, control directives, and operators. In the SASS pre-processor for instance, inheritance uses the @extend directive to share or extend the behaviour of an existing selector [1]. Control directives are the equivalent of the control-flow statement in object-oriented programming. These directives include @if, @for, @each, and @while statements [14] and are used for applying a style many times with variations [15]. Finally, CSS pre-processors have introduced operators include addition, subtraction, division, multiplication, relational operators and equality operators [14].

2.2. Structure of CSS

A basic CSS code is structured in a rule block or a rule set. A rule block is composed of a selector, opening brace, set of attributes and a closing brace [4]. Figure 1 demonstrates a basic building component of CSS code where P is a selector representing a paragraph and has a set of three attributes which determine the look of a paragraph. When this code is implemented, all the paragraphs of a Web page will have a background color of blue, the text font size will be of 12px and the text will be centered on the Web page.

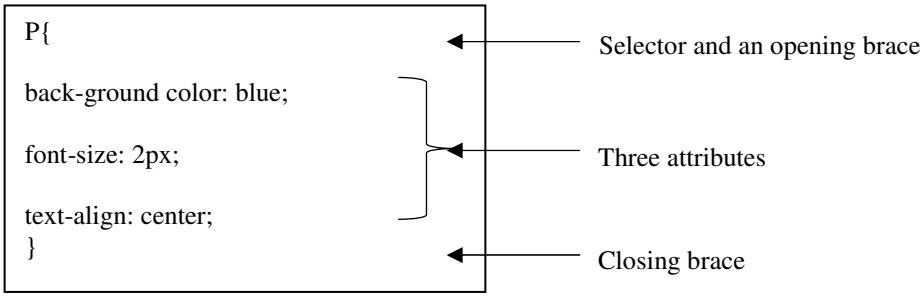


Figure 1. A sample CSS rule block

A careful analysis of both CSS and CSS pre-processors leads to the conclusion that there exist significant differences between them regarding the definition of functions, the use of mixins, inheritance, use of operators, use of control flow, nesting and declaration of variables. Table 1 shows the differences between CSS and CSS pre-processors.

Table 1. Comparison between CSS and CSS Pre-processors

Criteria	**CSS**	**CSS Pre-processors**
Functions definition	Only language defined functions	Language defined functions and programmer defined functions
Use of Mixins	No	Yes
Inheritance implementation	Inherits the values of parent element.	Selector Inheritance feature
Use of operators	No	Yes
Control flow statements	No control flows	Branch, loops, and calls
Nesting	No nesting	Nesting of rules
Declaration of variables	Must be declared within a rule block	Usually declared outside rule blocks

The demonstration of typical features of SASS pre-processor-based program is shown in figure 2. A variable named *color-accent* is declared and has been implemented in *h1* block, the mixin

block named *specs1* is defined and it's used in *h1* block by using @include statement, the use of inheritance is shown where *h2* block inherits *h1* block attributes by use of @extend directive, the use of control directive is demonstrated by use of @for statement) and finally use of function directive named *remy* which is called at *p* block.

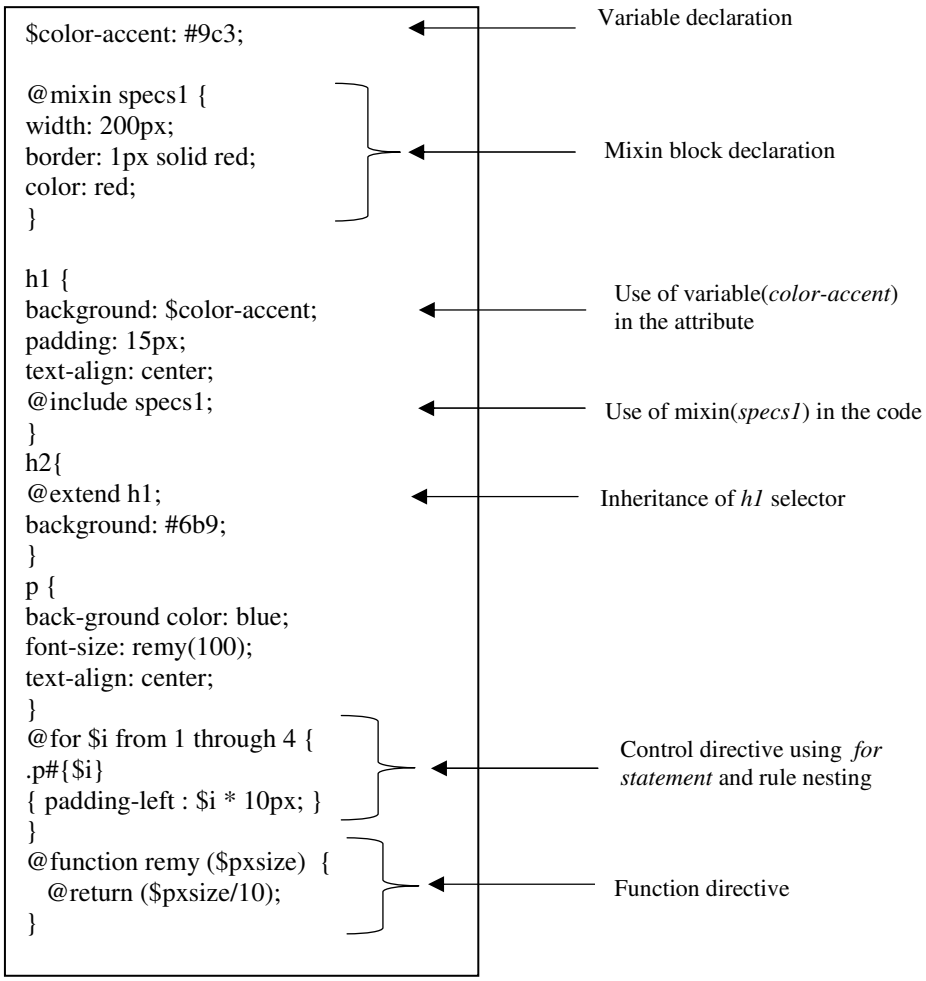


Figure 2. A sample SCSS program

3. Defining And Validating Metrics For Measuring CSS-Based Web Applications

Measurement of software is the process of defining, collecting and analyzing data regarding some software product to understand and control its complexity [16]. Software metrics are used to predict maintainability, reliability, and understandability of software [3, 4]. Three steps are required to define and validate software metrics, namely, the definition of new metrics, theoretical validation, and empirical validation [17].

3.1. Metrics Definition

Metrics definition is based on formal models such as the Goal Question Metric (GQM) [18], Balanced Scorecard (BSC) [19], and the Entity-Attribute-Metrics model (EAM) [20]. The GQM focuses on the organizational goals and has a wider scope which is at the project level [18] while the BSC which has its origin from strategic management, focuses on aspects of finance, clients,

internal, and learning and development [19]. The EAM model focuses directly on an object or entity such as a CSS module. Once the entity is identified, the next step is to identify attributes, and finally the definition of new metrics for measuring each attribute [17,20].

3.2. Theoretical Validation

Three popular theoretical validation frameworks that are cited in software metrics literature includes Weyuker's properties [21], Briand's framework [22] and Kaner framework [23]. These three frameworks have been used extensively by other metrics researchers to validate their metrics [4,6,8,9,24]. A summarized description of the three frameworks is given in the subsequent sections in CSS context.

3.2.1. Weyuker's Properties

Weyuker [21] proposed nine properties for validating software complexity metrics. In order to be able to analyse CSS metrics, Weyuker's properties have been re-written in the context of CSS language. A summary of these new CSS-based properties is shown below.

- Property 1 (Noncoarseness): $(\exists P)\ (\exists Q)\ (|P| \neq |Q|)$ where P and Q are two different CSS blocks. This property is satisfied when there exist two different CSS blocks P and Q such that |P| is not equal to |Q|, meaning they don't return similar metric results.
- Property 2 (Granularity): Let c be a non-negative number. Then there are finitely many CSS blocks of complexity c. This property asserts that if a CSS block changes then its complexity changes.
- Property 3 (Nonuniqueness): There can exist distinct CSS blocks P and Q where $|P| = |Q|$. This property affirms that two different CSS blocks can have the same metric value, this is to say that two CSS blocks have the same level of complexity.
- Property 4 (Design details are important): $(\exists P)\ (\exists Q)(P \equiv Q \ \&|P| \neq |Q|)$. There can be two CSS blocks P and Q whose external features look the same, however, due to different internal structure |P| is not equal to |Q|. This property asserts that two CSS blocks with the same number of attributes and directives could return different metric values.
- Property 5 (Monotonicity): $(\exists P)\ (\exists Q)\ (|P| \leq |P; Q| \ \&\ (|Q| \leq |P; Q|)$. This property asserts that if we concatenate two CSS blocks P and Q, the new metric value must be greater than or equal to the individual rule block.
- Property 6 (Nonequivalence of interaction): $(\exists P)\ (\exists Q)\ (\exists R)\ (|P| = |Q|$ and $|P; R| \neq |Q; R|)$ This property implies that if two CSS blocks have same metric value (P and Q), it doesn't necessarily mean that when each of the CSS blocks is concatenated with similar CSS block R, the resulting metric values are the same.
- Property 7 (Permutation): If you have two CSS blocks P and Q which have the same number of attributes in a permuted order, then |P| is not equal to |Q|.
- Property 8 (Renaming property): if P is assigned as Q, then $|P| = |Q|$. Where you have two CSS blocks P and Q differing only in their selector names, then |P| is equal to |Q|.
- Property 9 (Interaction increases complexity): $(\exists P)\ (\exists Q)\ (|P| + |Q| < (|P; Q|)$. This property asserts that there exist two CSS blocks P and Q, where the complexity metric value of the two CSS blocks when summed up is less than when the rule blocks are interacting.

3.2.2. Briand's Framework

Briand et al. [22] proposed this property-based approach for software measurement. This approach formalizes software attributes into size, length, complexity, cohesion, and coupling. In each of the attribute, there are properties that should be satisfied for any metric falling under it.

Size: The size of the code C is a function size(C) characterized by the following three properties namely; non-negativity, null value and module additivity which should be satisfied by the size metrics.

- Property 1 (Non-negativity): the size of code must never be negative i.e., size (C) ≥ 0.
- Property 2 (Null values): the size of the code is null if there is no block i.e. size (C) =0.
- Property 3 (Module additivity): the code size is the summation of two blocks (B1 and B2) i.e. Size (C) = size (B1) + size (B2).
- **Length:** The length of the code C is a function length (C) characterized by the following five properties namely; Non-negativity, null value, disjoint modules, non-increasing monotonicity, and non-decreasing monotonicity.
- Property 1 (Non-negativity): The length of the code cannot be negative.
- Property 2 (Null value): The length of the code is null if the code has no CSS blocks.
- Property 3 (Disjoint modules): The length of a code that has two separate blocks is equal to the lengths of the two CSS blocks.
- Property 4 (Non-increasing monotonicity): Adding relation between elements of a CSS block does not increase the length of the code.
- Property 5 (Non- decreasing monotonicity): Adding relation from two CSS blocks does not decrease the length of code.

Complexity: The complexity of code C is a function complexity (C) that is characterized by the following five properties namely; Non-negativity, null value, disjoint module additivity, symmetry and module monotonicity.

- Property 1 (Non-negativity): The complexity of the code cannot be negative.
- Property 2 (Null value): The complexity of the code is null if the block is empty.
- Property 3 (Disjoint module additivity): The complexity of the code that has two CSS blocks is the summation of the complexities of the two blocks.
- Property 4 (Symmetry): The complexity of a code is not dependent on how you choose to represent code elements relationships.
- Property 5 (Module monotonicity): The complexity of a code is no less than the sum of the complexities of any two of its CSS blocks with no relationships in common.

Cohesion: The cohesion of the code C is a function cohesion (C) characterized by the following four properties namely; Non-negativity and normalization, null value, monotonicity, and cohesive modules.

- Property 1 (Non-negativity and normalization): The cohesion of the code cannot be negative, and the measure should be independent of the size of the CSS block.
- Property 2 (Null value): The cohesion of the code is null if the CSS block is empty.
- Property 3 (Monotonicity): The relationship between CSS blocks cannot decrease cohesion.
- Property 4 (Cohesive modules): The relationship between CSS blocks cannot decrease cohesion when two blocks showing no relationship are encapsulated.

Coupling: The coupling of code C is a function coupling (C) that is characterized by the following five properties namely; Non-negativity, null value, disjoint module additivity, merging of modules and monotonicity.

- Property 1 (Non-negativity): The coupling of code cannot be negative.

- Property 2 (Null value): The coupling of the code is null if there is no internal relation between the CSS blocks.
- Property 3 (Disjoint module additivity): The coupling of the code increases when more CSS blocks are added that share global data.
- Property 4 (Merging of modules): The coupling of the code decreases when two CSS blocks are merged.
- Property 5 (Monotonicity): The coupling of the code increases when the relationship between CSS blocks increases.

3.2.3. Kaner's Framework

This framework is more practical than the formal approach of Weyuker's properties and Briand's framework [9]. The Kaner framework [23] evaluates software metrics to establish the purpose of the defined measure, scope of the measure, the attributes to measure, natural scale of the attributes to measure, natural variability of the attribute, metrics defined, measuring instrument, natural scale for the metric, natural variability of readings, relationship of attribute to the metric value and the natural and foreseeable side effects based on use of the instrument.

3.3. Empirical Validation

Metrics researchers frequently employ experiments, case studies, or surveys in their effort to validate their new metrics [17, 25]. Empirical validation is conducted to establish the usefulness of new metrics by the industry [17].

3.3.1. Experiments

Out of these three empirical strategies, experimentation is the more frequently used due to its formal, rigorous and repeatable characteristics [17, 26]. The subjects are randomly assigned different treatments for the purpose of keeping one or more variables constant and other variables are manipulated. The effects of variable manipulation are observed, measured and interpreted [17].

3.3.2. Case studies

Case studies involve closer and deeper study on an attribute or relationship between several attributes. The context in which the attributes under study are being observed is an important factor in case studies [26].

3.3.3. Surveys

Surveys are regarded as a retrospective study where you study a situation and unlike experiments and case studies the variables cannot be manipulated [20]. In surveys an existing tool, method or technique is investigated by taking a sample from a population, the results generated from the sample are analyzed and can be generalized to the population [26].

3.4. Metrics Tool Support

Metric tools automate software metrics collection and calculation, thus serving as useful component of the quality control process [17]. Metrics tool basically extract required units from the code and provide metrics value. Static metrics tool mainly consists of three components, lexical analyzer, parser and an output component. The lexical analyzer accepts source code and

breaks it into tokens. The parser counts the tokens and computes the metrics while the output component is used to display the computed metrics [27].

Some of the existing metrics tool are such as Analyst4j which analyzes quality of java programs, CCCC an open source command-line tool, Chidamber and Kemerer Java metrics tool and VizzAnalyzer which performs quality analysis on software code [28]. In style sheets CSS analyzer tool was developed to measure different metrics for CSS [29]. This tool was evaluated and found to be working in far less time than the manual process.

Tools provide the insight concerning the quality of source code and are a basic requirement for acceptability of software metrics proposed in the software industry [28, 29].

4. EXISTING COMPLEXITY METRICS

There are several researchers who have defined metrics in web domain. This section describes the different metrics based on the existing web-based languages. A summary of whether there is tool support, and whether theoretical validation and empirical validation is conducted has been indicated. The metrics have been categorized by author name.

4.1. Misra and Cafer metrics

Misra and Cafer [30] proposed JavaScript Cognitive Complexity Measure (JCCM), for measuring the design quality of scripts. The motivation for JCCM is to calculate the structural and cognitive complexity of JavaScript. This metric considered five factors that contribute to JavaScript complexity namely; the number of lines of codes, number of meaningfully named variables (MNV), number of arbitrary named distinct variables (ANDV), the cognitive weight of basic control structures (BCS's) and number of operators. JCCM metric has been successfully theoretically validated via measurement theory and empirically validated by analyzing thirty JavaScript files. However, there is no indication of tool support, meaning its difficult for the industry and researchers to adopt it.

4.2. Basci and Misra metrics

Basci and Misra [31] defined an entropy measure for the assessment of structural complexity of XML. The schema entropy metric measures the schema documents complexity due to elements structure diversity. This metric has been validated empirically although there is no evidence of theoretical validation and tool support.

Basci and Misra [32] defined two document type definition (DTD) complexity metrics, Entropy metric: E-DTD and Distinct Structured Element Repetition Scale metric: DSERS-DTD so as to measure the structural complexity of schemas in DTD language. E(DTD) metric value is computed by considering equivalence classes in a schema document. An equivalence class is the one that its elements have the same value of fan-in and fan-out and number of attributes. DSERS(DTD) metric measures the interface complexity of the schema document. The lower the E metric value and the higher the DSERS value the lesser the effort to understand the element structure. These metrics have been theoretically and empirically validated, although no support for the automated tool has been seen so far.

Basci and Misra [33] have also defined a design complexity metric for XML Schema documents (XSD) written in W3X XML Schema language. The metric C(XSD) measures the complexity of XSD based on the internal architecture of XSD components and recursion. It captures all the

major factors responsible for XSD complexity. These factors are complexity based on elements and attributes definitions, elements and attributes group definitions, user-defined or built-in simple type and complex type definitions, elements definitions with no recursion and components that are included from external schema files. The proposed metric has been validated both through an experiment and theoretically through the Kaner and Briand's framework. Tool support for this metric has however not been seen although desirable.

In [10], Basci and Misra, four XML web service metrics are defined namely data weight of a web service description language (DW -WSDL) which is computed by defining the sum of the data complexities of each input and output messages, distinct message ratio (DMR) metric which counts the number of distinct structured messages, message entropy (ME) metric which measures the complexity of similar-structured messages and message repetition scale (MRS) metric analyses the varieties in structures of web service description language. These four metrics have been theoretically validated using Kaner Framework and Weyuker's properties. They have also been validated empirically although there is no automated tool support.

4.3. Thaw and Misra metrics

Thaw and Misra[34] have defined an Entropy Measure of Complexity (EMC) to measure the reusable quality of XML schema documents. When EMC value is high it implies that the document is more reusable and it contains inheritance feature, elements, and attributes. This metric was validated theoretically using Kaner framework and Weyuker's properties and empirically through an experiment. As is the case with most metrics in this domain, no tool support has been seen for the EMC metric.

4.4. Tamayo, Granell and Huerta metrics

Tamayo et al.[35] defined three XML complexity metrics in geospatial web services namely Data Polymorphism Rate (DPR), Data Polymorphism Factor (DPF), and Schemas Reachability Rate (SRR). DPR measures schema polymorphism, DPF measures the influence of polymorphic elements in the overall schema complexity, and SRR measures the fraction of imported hidden schema components by the subtyping mechanisms. These metrics have been empirically validated using a case study and were found to be useful in detecting potential design problems for-example a component with too many information items. The metrics have not been theoretically validated and there is no evidence of tool support.

4.5. Adewumi, Misra and Ikhu-Omoregbe Metrics

Unlike other metrics described in previous sections, these metrics have a direct focus on CSS. Adewumi et al.[4] have proposed six CSS metrics namely, Rule Length, Number of Rule Blocks, Entropy Metric, Number of Extended Rule Blocks, Number of Attributes defined per Rule Block, and Number of Cohesive Rule Blocks.

The Rule Length metric measures the number of lines of rules (or code) in a CSS file, and its intended to measure the size of code. It is adapted from the popular line of code (LOC) metric. The formula for calculating rule length is

$$RL = \sum \text{rule statements}$$

Where RL is the rule length, and rule statements are the number of executable statements in a CSS file.

The limitation with RL metric is that it doesn't put into consideration the non-executable parts of CSS code such as white spaces or comment lines.

The Number of Rule Blocks metric4 counts the number of rule blocks in CSS code. The formula for calculating the Number of Rule Blocks is:

NORB=∑ rule blocks in a CSS file

Where NORB is the Number of Rule Blocks in CSS file, and a rule block consists of a selector and its declarations.

This metric is like the RL metric because its intended to measure the size of the code, meaning it achieves the same goal as RL.

The Entropy Metric puts the elements with the same structural complexity in the same category, this category is referred to as equivalence class(C). The entropy of a CSS document is based on n distinct class of elements and is calculated using the relative frequencies as unbiased estimates of their probabilities. P (Ci), i=1, 2…. n. The formula for calculating the entropy of CSS file is:

$$E=\sum P(Ct)\log_2 P(Ct) \text{ where } t=1\ldots n$$

$$=\sum (1/n)\log_2(1/n)$$

Where E represents the Entropy metric value, P represents probability of occurrence of distinct class elements (C).

This metric basically groups similar rule blocks and so when the entropy metric value is low the higher the structural similarity of rule blocks meaning the complexity of the CSS code is low.
The Number of Extended Rule Blocks metric counts the number of rule blocks that have similar selector name in a CSS file. The formula for calculating the Number of Extended Rule Blocks of CSS file is:

NERB=∑ extended rule block(i)

where NERB represents the Number of Extended Rule Block and i = 1…. n

The Number of Attributes defined per Rule Block metric determines the average number of attributes defined in the rule blocks of a CSS file. The formula for calculating the Number of Attributes defined per Rule Block of CSS file is:

NADRB= (Total number of attributes in all rule blocks / Total number of rule blocks)

Where NADRB represents The Number of Attributes defined per Rule Block

The higher the NADRB metric value the higher the complexity of the CSS code.

The Number of Cohesive Rule Block metric counts all rule blocks possessing a single attribute. The formula for calculating the Number of Cohesive Rule Block of CSS file is:

NCRB=∑ rule block (i) with a single attribute

Where NCRB represents the Number of Cohesive Rule Blocks and i = 1 …. n

The higher the NCRB metric value the lower the complexity of CSS code.

These CSS complexity metrics have been evaluated and found to satisfy the Kaner framework. However, these metrics have not been empirically validated and no effort to validate the metrics with the more widely used Weyuker's properties or even Briand's Framework has been conducted by the authors. A tool to support the collection and computation of CSS metrics have been developed [28]. However, its accuracy level has been measured at 51.25% [28], meaning that it cannot be relied on fully.

5. CONCLUSIONS AND FUTURE WORK

This study shows that several metrics have been proposed in the web domain and most of the metrics shown were empirically validated, however there still no rigorous theoretical validation. A fully theoretically validated metric should be validated using a practical framework such as Kaner framework and use of measurement theory such as Weyuker's properties and Briand's framework to test the mathematical soundness of a metric. In addition, there is no evidence of automated tool. The automation of metrics collection and computation enhances the acceptability of the metrics by the industry. This implies that there is a need for researchers to put an effort to ensure that the metrics defined are theoretically sound, they measure what they intended and that they increase industry acceptance of the new metrics. The existing CSS complexity metrics which are the focus of this study cannot be relied upon since we cannot tell their mathematical soundness and no empirical validation has taken place. These CSS metrics don't take into consideration the extra features of CSS pre-processors, meaning their applicability in measurement and control of CSS pre-processors complexity is put to question. It's also notable that the existing tool for CSS metrics computation is deficient in its effectiveness because it only achieves 51.25% accuracy.

Considering the above issues raised, it means that efforts should be made to define new complexity metrics for CSS and CSS pre-processors and then validate them theoretically and empirically. A metrics tool that is highly effective should also be given priority in order to convert the proposed metrics into viable industry solutions.

REFERENCES

[1] Mazinanian, D., & Tsantalis, N. (2016) An empirical study on the use of CSS preprocessors. In 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Vol. 1, pp168-178.

[2] Wolf, D., & Henley, A.J., (2017) Cascading Style Sheets (CSS). In Java EE Web Application Primer. Apress, Berkeley, CA, pp115-118.

[3] Ogheneovo, E. E., (2014) On the Relationship between Software Complexity and Maintenance Costs. Journal of Computer and Communications, Vol 2, No. 14, pp1-16.

[4] Adewumi, A., Misra, S., & Ikhu-Omoregbe, N., (2012) Complexity metrics for cascading style sheets. In International Conference on Computational Science and Its Applications, Berlin, Heidelberg., pp248-257.

[5] Muketha, G.M., Ghani, A.A.A., Selamat, M.H. & Atan, R, (2010b) Complexity Metrics for Executable Business Processes. Information Technology Journal, Vol. 9, No. 7, pp1317-1326.

[6] McCabe, T. J., (1976) A complexity measure. IEEE Transactions on software engineering, Vol. 4, pp308-320.

[7] Chidamber, S. R., & Kemerer, C. F., (1994) A metrics suite for object-oriented design. IEEE Transactions on Software Engineering, Vol. 20, No. 6, pp467-493.

[8] Misra, S. , Adewumi, A. , Fernandez-Sanz, L. & Damasevicius, R .,(2018) A Suite of Object Oriented Cognitive Complexity Metrics. IEEE Access, Vol. 6, pp8782-8796.

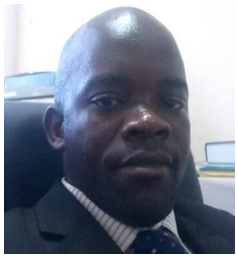
[9] Pichler, C., Strommer, M., & Huemer, C., (2010) Size matters!? measuring the complexity of xml schema mapping models. In 2010 6th World Congress on Services, IEEE. pp497-502

[10] Basci, D., & Misra, S., (2011a) Metrics Suite for Maintainability of XML Web-Services. IET Software, Vol. 5, No. 3, pp320-341.
[11] Hissom, A., (2011) Introduction to HTML5 and CSS3. Available: http://amyhissom.com/HTML5-CSS3/history.html
[12] Lie, H. W., & Bos, B., (2005) Cascading Style Sheets: Designing for the Web (3rd ed.). Boston, MA, USA: Addison-Wesley Professional.
[13] Charpentier, A., Falleri, J. R., & Réveillère, L., (2016) October. Automated Extraction of Mixins in Cascading Style Sheets. In IEEE International Conference on Software Maintenance and Evolution (ICSME). pp56-66.
[14] Henley, C., (2015) Better CSS with Sass. UK: Five Simple Steps. ISBN: 978-3-863730-81-9
[15] Catlin, H., & Catlin, M. L.,(2011) Pragmatic Guide to Sass. (K. Keppler, Ed.) USA: The Pragmatic Programmers, LLC.
[16] McGarry, J., Card, D., & Jones, C., et al., (2002) Practical Software Measurement: Objective information for decision makers. Boston, USA: Addison Wesley.
[17] Muketha, G. M., Ghani, A. A. A., Selamat, M. H., & Atan, R, (2010a) A Survey of Business Process Complexity Metrics. Information Technology Journal, Vol 9, No. 7, pp1336-1344.
[18] Basili, V. R.,(1992) Software modeling and measurement: the Goal/Question/Metric paradigm.
[19] Martinsons, M., Davison, R., & Tse, D.,(1999) The balanced scorecard: a foundation for the strategic management of information systems. Decision support systems, Vol 25, No.1, pp71-88.
[20] Fenton, N. & Pfleeger, S.L, (1997) Software Metrics: A Rigorous and Practical Approach, 2nd Edition, IT Publishing Company.
[21] Weyuker, E. J.,(1988) Evaluating software complexity measure. IEEE Transaction on Software Engineering, 14(9): pp1357-1365.
[22] Briand, L.C., Morasca, S., & Basili, V.R., (1996) "Property-Based Software Engineering Measurement".
[23] Kaner, C.,(2004) Software Engineering Metrics:what do they measure and how do we know? In:Proc. Tenth Int. Software Metrics Symp. pp1-10.
[24] Geneves, P., Layaida, N., & Quint, V.,(2012) On the analysis of cascading style sheets. In proceedings of the 21st international conference on World wide web. ACM. pp809-818
[25] Srinivasan, K.P. & Devi, T., (2014). Software metrics validation methodologies in software engineering. International Journal of Software Engineering & Applications (IJSEA), Vol. 5, No. 6, pp87-102.
[26] Wohlin, C., Runeson, P., Höst, M., Olsson, M.C., Regnell, B. & Wesslén, A., (2000) Experimentation in Software Engineering: An Introduction, Kluwer Academic Publishers.
[27] Scotto, M., Sillitti, A., Succi, G., & Vernazza, T., (2004) Dealing with Software Metrics Collection and Analysis: a Relational Approach. Stud. Inform. Univ., Vol. 3, No. 3, pp343-366.
[28] Lincke, R., Lundberg, J., & Löwe, W., (2008) Comparing software metrics tool. In ACM proceedings of the 2008 international symposium on Software testing and analysis. pp131-142.
[29] Adewumi, A., Emebo, O., Misra, S., & Fernandez, L.,(2015) Tool Support for Cascading Style sheets' Complexity Metrics. Communications in Computer and Information Science. pp551-560.
[30] Misra, S., & Cafer, F., (2012) Estimating Quality of JavaScript. The International Arab Journal of Information Technology, Vol. 9, No.6, pp535-543.
[31] Basci, D., & Misra, S., (2011b) Entropy as a measure of quality of XML schema document. The International Arab Journal of Information Technology, Vol 8, No.1, pp75-83.
[32] Basci, D., & Misra, S.,(2011c) Document Type Definition (DTD) Metrics. Romanian Journal of Information Science and Technology, Vol 14, No.1, pp31-50.
[33] Basci, D., & Misra, S.,(2009) Measuring and evaluating a design complexity metric for XML schema documents. Journal of Information Science and Engineering, Vol 25. pp1405-1425.
[34] Thaw, T., & Misra, S. (2013) Measuring the Reusable Quality for XML Schema Documents. Acta Polytechnica Hungarica, Vol. 10, No.4, pp87-106.
[35] Tamayo, A., Granell, C., & Huerta, J., (2011) Analysing complexity of XML schemas in geospatial web services. In Proceedings of the 2nd international conference on computing for geospatial research & applications. ACM. pp17.

AUTHORS

John Gichuki Ndia is a Tutorial Fellow at the Department of Information Technology at Murang'a University of Technology, Kenya. He earned his Bachelor of Information Technology from Busoga University in 2009, and his MSc. in Data Communications from KCA-University in 2013. He is currently pursuing the PhD in Information Technology at Masinde Muliro University of Science and Technology. His research interests include Software quality, software metrics and network security. He is an IEEE member and a member of the International Association of Engineers (IAENG) society of Software Engineering.

Geoffrey Muchiri Muketha is Associate Professor and Dean of the School of Computing and Information Technology, Murang'a University of Technology, Kenya. He received his BSc. in Information Science from Moi University in 1995, his MSc. in Computer Science from Periyar University in 2004, and his PhD in Software Engineering from Universiti Putra Malaysia in 2011. He has many years of experience in teaching and supervision of postgraduate students. His research interests include software and business process metrics, software quality, verification and validation, empirical methods in software engineering, and component-based software engineering. He is a member of the International Association of Engineers (IAENG).



Kelvin Kabeti Omieno is a Senior Lecturer and Dean, School of Computing and Information Technology, Kaimosi Friends University College, Kenya, A Constituent College of Masinde Muliro University of Science and Technology. He holds a PhD in Business Information Systems of Jaramogi Oginga Odinga University of Science & Technology. He has MSc in Information Technology and Bachelor of Science in Computer Science from Masinde Muliro University of Science and Technology. He has been involved in several research projects of ICTs for Development, Data Analytics, Computational Grid, Machine Learning, Health Informatics, E-learning systems and E-waste management in Kenya. Besides, he has published widely in journals and conference proceedings in Information technology and ICTs for development. He is a professional member of the Association for Computing Machinery (ACM), the largest association of computing professionals globally and is a reviewer with two International Journals.