STRUCTURAL COMPLEXITY ATTRIBUTE CLASSIFICATION FRAMEWORK (SCACF) FOR SASSY CASCADING STYLE SHEETS

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

[1]School of Computing and Information Technology,
Murang'a University of Technology, Kenya
[2]School of Computing and Information Technology,
Kaimosi Friends University College, Kenya

ABSTRACT

Several researchers have proposed the various classes of software attributes to guide in the derivation of metrics for software products. These existing classifications have targeted traditional software paradigms such as procedural and object-oriented software. Sassy cascading style sheets (SCSS) has unique features since it combines Cascading style sheets (CSS) features with traditional software features such as variables, functions and control flows. Due to this uniqueness, there arises a need to develop a new classification scheme that can be effectively used to classify all the possible structural attributes for Sassy cascading style sheets. The aim of this paper, therefore, is to develop and validate a comprehensive software complexity attributes classification framework for SCSS. The new framework was validated through an online expert opinion survey, where thirteen SCSS experts were involved. Results show that the proposed framework is complete and effective to guide metrics researchers in defining new metrics for SCSS

KEYWORDS

Cascading Style Sheets, SCSS Complexity classification framework., Software attributes, Structural complexity

1. INTRODUCTION

The practice of defining software metrics has been continuing over the years for different kinds of software domains such as procedural, object-oriented, and web-based domains among others. These metrics are based on software attributes, for-example the popular McCabe's Cyclomatic Complexity metric is based on the control flow attribute of software [1], while some of the Chidamber and Kemerer metrics such as the Depth of Inheritance Tree (DIT) and Number of Children (NOC) are based on inheritance attribute [2]. Therefore, it is prudent to first identify the attributes to be measured for that software before attempting to derive new metrics. The software attribute is defined as the feature or property of a product [3].

Fenton and Bieman [5] in effort to create an industry standard for determining the process of defining metrics have identified three major stages, including identification of entity to measure (e.g. project, product and process), identification of the entity's attributes that need to be measured, and then deriving metrics for each of the attributes. Several researchers have proposed classification schemes for software attributes to aid metrics definition [4]-[11].

Some of the existing software attributes classification schemes provide a general treatment of complexity [4]-[6] while others focus more on structural complexity [7]-[8]. While Daudi and Kadir [7] classified complexity attributes for service-oriented architecture (SOA) and Muketha

[8] classified complexity attributes for business process models, there has been little effort to classify structural complexity attributes for the stylesheets' domain.

Sassy Cascading Style Sheets (SCSS) is an extension of Cascading Style Sheets (CSS) and it combines CSS features and traditional software features such as the use of variables, mixins, functions, and control flows [12]. This uniqueness of SCSS software means that the existing classification schemes cannot be used to sufficiently identify the structural attributes for SCSS.

The methodology employed in this study was to first identify existing classification schemes, their limitations, and then extend one of them to come up with a classification scheme for SCSS. Muketha's classification [8], was adopted for the extension as it is the most closely related to this study. An online expert's opinion survey was conducted to collect data, and the data were analyzed using descriptive statistics to validate the proposed framework.

The rest of this paper is structured as follows. Section two presents the existing classification schemes, section three presents structural complexity, section four presents the new classification framework for SCSS structural complexity, section five presents validation results, and section six presents the conclusions and future work.

2. EXISTING CLASSIFICATION SCHEMES

Several studies have attempted to classify software complexity attributes and are therefore closely related to the work presented in this paper. Fenton and Pfleeger [5] and Fenton and Bieman [4], proposed three categories for deriving the attributes to measure namely; process, product, and resources. The product category which is the focus of this study further classified attributes as internal or external attributes. Internal attributes are those that can be measured directly such as the size of code while external attributes are measured indirectly, such as reliability and maintainability. The limitation of this classification is that the modularity of the attributes such as control flow, data flow, cohesion, and coupling is not known.

In another study [5] they identified four ways of categorizing software attributes into the product, process, people, and value to the customer. In this classification scheme, structural complexity falls under the product category. Structural complexity is further divided into control flow complexity, data complexity, and size attributes. The limitation of this classifications scheme is like the Fenton and Bieman classification, in that, the level of modularity of the attributes is not provided, meaning we can't tell whether all the possible attributes of software are captured.

Daud and Kadir [7] have classified software structural attributes into static and dynamic attributes. These authors identified three structural attributes, coupling, cohesion and complexity which fall under both static and dynamic. These attributes are the most popular in measuring service-oriented architecture (SOA). The limitation of this classification is that it identified the attributes from the literature and not from the structural properties of SOA. Meaning that the attributes identified may not fully represent SOA structural complexity.

Mens [10] identified four major dimensions of software complexity, including theoretical complexity, the complexity of use, organizational complexity and structural complexity. Theoretical complexity was further divided into computational and algorithmic complexity, complexity of use was divided into functional and usability, while structural complexity was divided into module level and system level. This classification scheme does not show what attributes can be derived from module level and system level hence it's not comprehensive.

Henderson-Sellers [11] categorized software complexity into computational complexity, psychological complexity, and representational complexity. The author further divided psychological complexity into structural complexity, programmer characteristics and problem complexity. Structural complexity was further divided into intra and inter-module categories. The intra-module category is further divided into size, control flow, and cohesion attributes while the inter-module category is specialized into the coupling attribute. This classification scheme is limited in that it overlooks some new dimensions of structural complexity evident in SCSS software.

The part of structural complexity in the Henderson-Sellers classification scheme has been extended by introducing the hybrid category to the existing inter and intra-module categories [8]. The hybrid attribute category combines features from intra-module and inter-module attributes. Muketha's work is limited in that it overlooks some new dimensions of structural complexity introduced in SCSS software. However, this study extended Muketha's framework because it's more recent and comprehensive in the context of structural complexity. Figure 1 illustrates the classification framework. The inter-module attributes focused on an individual process which is equivalent to a module, inter-module attributes focused on the interaction of two process modules while hybrid attributes combine the features of both intra-module and inter-module attributes.

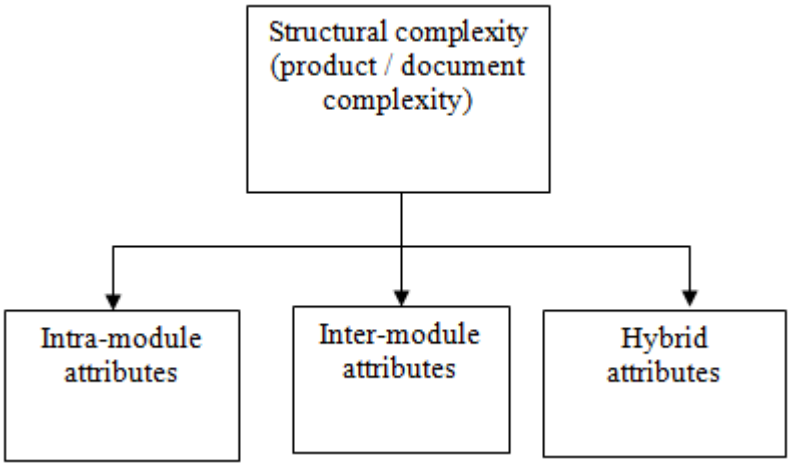


Figure 1. Structural complexity attributes classification [8]

3. STRUCTURAL COMPLEXITY

Structural complexity is defined as how the program elements are organized and interact within the software system [12], [13]. It is concerned with the measurement of internal attributes and is assessed by the difficulty of performance of tasks such as the writing of codes, modifying and testing of software [10], [14]. The identification of the right attributes for a given software can help in the evaluation and improvement of a software product [15].

3.1. Structural Complexity Properties for Software

Many authors consider size, length, coupling, and cohesion as part of structural complexity [8], [11], [16]. For instance, the lines of code (LOC) metric, also called the physical lines of code, has been used as a size measure, and to some extent, as a complexity measure. The related logical lines of code (LLOC) metric, has been found to have higher accuracy when compared to LOC because it eliminates comment lines, auto-generated code lines, header files, ineffective code lines, compiler directives, labels, and empty case statements [16]. For example, Adewumi *et al.*

[17] proposed size in terms of lines of rules for cascading style sheets while Misra and Cafer [18] considered size in terms of lines of JavaScript code on condition that the only lines to be factored were those that consisted of variable(s) or operators.

The concept of inheritance has been recognized as one of the most important features of software reuse. In object-oriented languages, inheritance supports class hierarchy design and captures the is-a relationship between a class and sub-class [19]. Inheritance has been studied in object-oriented languages extensively [19]-[22]. Though inheritance supports reuse, it can increase complexity if not used in the proper range [21]. Style sheets provide a unique way of supporting inheritance because there are no classes and sub-classes as provided for in the object-oriented domain.

Nesting complexity has also been studied as an important property. Nesting reflects the level of nesting within constructs or control structures [23]. Constructs are such as if, case, for, while, and do-until can be nested. A statement that is at the innermost level is harder to understand, meaning that it contributes more to complexity than other statements [24]. In SCSS, nesting occurs with selectors, that is, the more the selectors are deeply nested the more complex an SCSS code becomes [25].

Coupling has been defined as the measure of the strength of association established by a connection from one module to another [26]. It has been argued that the stronger the coupling between modules, the more difficult these modules are to understand, change and correct, resulting in more complex software. Coupling has been studied in the domain of procedural programming [26] and object-oriented programming [2], [27], [28]. While coupling as a complexity measure has been studied in procedural and object-oriented languages it has not been addressed in the stylesheets' domain.

The aspect of cohesion is discussed extensively in the procedural and object-oriented domain. Cohesion is defined as the 'single-mindedness' or 'relatedness' of a module component [29]. When a module is highly cohesive, it means, all the defined elements in a module perform a single task. Therefore, it's the goal of software designers to make a program as cohesive as possible.

The Complexity of code can be expressed through control structures, and therefore, a program which implements control structures is regarded as more complex in comparison to the program without control structures [24]. The complexity of a program is directly proportional to the cognitive weights of Basic Control Structures [18]. For-example, iterative control structures like for loop, while, and do…while are more complex than decision making control structures such as if…then…else.

3.2. Structural Properties for SCSS

SCSS is a web-based language that is implemented in Syntactically Awesome Style Sheets (SASS) pre-processor. Its purpose is to style web documents written in Hypertext mark-up language (HTML) and Extensible mark-up language (XML) [30]. SCSS combines the characteristics of CSS, such as the use of selectors, rule blocks, and declarations with those of traditional software such as inheritance, nesting, and coupling [30]. The combination of these features makes the front web developers create more efficient and maintainable code.

SCSS provides a unique way of supporting inheritance through selector inheritance. The selectors are extended in an SCSS rule block by use of @extend directive. This means that all the attributes of the inherited selector are implemented in the rule block that the selector has been extended.

Figure 2 has a code that illustrates the use of selector inheritance. The code has two rule block which has a selector named .alarm and is inherited by .alarm-positive selector. This means that the .alarm-positive selector will have five attributes or declarations i.e. padding, font size, text align, color and background.

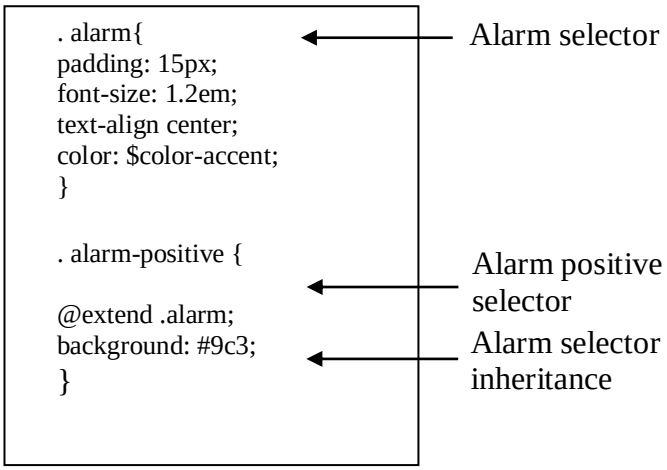


Figure 2. Selector inheritance

SCSS allows nesting of rules inside each other instead of repeating selectors in separate declaration [31]. Figure 3 illustrates nesting by placing the message rule block inside infobox rule block.

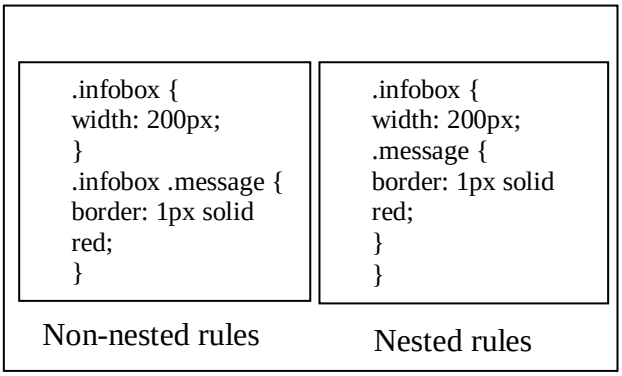


Figure 3. Nesting of rules

SCSS consists of rule blocks, a rule block consists of properties and values which together form a declaration or an attribute. The more the number of components defined in a CSS rule block, the more complex it is [17]. SCSS has several components that contribute to rule block complexity, for-example, attributes or declarations, operators, variables, function calls, control directives, include directive and extend directive.

In SCSS, coupling is manifested when the declared properties such as mixins and variables are used in several places of the code, meaning that the properties can be changed without realizing you are affecting multiple objects at once or not noticing which elements are being affected by the changes.

In stylesheets, cohesion is viewed as the rule blocks having a single attribute [17]. The more the SCSS rule blocks with a single attribute, the lesser its complexity, thus increasing the maintainability of the code.

4. A NEW CLASSIFICATION FRAMEWORK FOR SCSS STRUCTURAL COMPLEXITY

The proposed classification framework extends the work of Muketha [8] with the incorporation of new attributes found in SCSS.

4.1. Architecture of the Proposed Framework

In the proposed framework, intra and inter-module, as well as hybrid attributes, have been redefined and re-interpreted, and a new category called extra-module attribute added.

In the context of SCSS, the intra-module focuses on attributes that can be derived from a single rule-block which is equivalent to a module. Two main categories were identified, size and control flow complexity. In the size category, the features that can be used to determine SCSS code size are identified namely the number of declarations or attributes, number of operators and number of rule blocks. In order to determine the control-flow complexity of SCSS, control directives i.e. @for, @if, @each, etc. must be identified in the code.

Inter-module in SCSS focuses on the interaction of the various rule-blocks. In the proposed framework, the inter-module has been divided into inheritance complexity and nesting complexity categories. Inheritance complexity in SCSS happens when the styles or values are shared by using extend directive, this is known as selector inheritance. SCSS nesting complexity occurs when the rules are put inside each other.

The hybrid attribute combines features of at least two categories of structural complexity, for-example, intra-module and inter-module [8]. In SCSS the hybrid attribute has one category named association complexity. This kind of complexity is led to by different features, found in different categories of SCSS structural complexity being implemented in a single rule block. For-example, the sharing of variables and mixins by rule blocks leads to information flow complexity, while the use of extend directive in a rule block leads to inheritance complexity. Information flow complexity falls in the extra-module attribute category while inheritance complexity falls under inter-attribute complexity. The combination of these two categories leads to a hybrid attribute. In the framework, the example given under association complexity has @extend (derived from the inter-module category) and @include (derived from extra-module category).

Extra-module attribute focuses on the interaction of modules via an external module. In SCSS the Extra-module attribute focuses on rule-blocks interacting with mixins and/or global variables. These mixins and global variables are defined outside of SCSS rule blocks. When several rule blocks are sharing the same mixin and global variable, then the rule blocks are deemed to be coupled with each other.

This implies that a change in the values of a mixin and a variable will affect all the rule blocks that are sharing the mixin and global variable. Figure 4 below illustrates the proposed structural complexity attribute classification framework for SCSS.

4.2. Application of the Framework

This section aims to providing an interpretation of the proposed framework through a real-life scenario (Appendix).

The intra-module attribute is the first category of the SCSS structural complexity, and it considers complexity in terms of size and control flow complexity. The size of the SCSS file can be determined based on the number of attributes, number of operators or number of rule blocks. For example, to determine the size of the file provided in the Appendix based on the number of rule blocks, count all the rule blocks, where each rule block is recognized by an opening brace ({) and a closing brace (}). The control flow complexity of SCSS code is determined by the control directives implemented in the code. In the SCSS code provided, the @for directive has been implemented, meaning that the measurement for the control flow complexity can be determined.

The inter-module attribute category has described inheritance and nesting complexity. Inheritance complexity in SCSS is introduced by the use of @extend directive. In the file provided, the extend directive has been used in h2 element selector to inherit p element selector. The nesting complexity in SCSS considers nesting of rules. In the code provided the @media directive has modal dialog class selector inside it.

In the hybrid attribute category, a form of complexity known as association complexity is identified. In the SCSS code provided this kind of complexity is demonstrated in the *p* element rule block. To determine the complexity of *p* rule block the number of attributes that fall under the intra-module category are identified. In the same rule block, there is the use of mixin (*PlayfairDisplay-Regular*) and variable (*color1*) which leads to coupling, meaning that the extra-module category has been used. Lastly, the extend directive has been used in the *p* rule block, which introduces inheritance complexity under the inter-module category.

The final category known as the extra-module category is illustrated. The information flow complexity which is a result of coupling through the use of mixins and global variables is demonstrated in the SCSS code provided. The *span*, *h3*, and *h4* element selectors make use of a mixin named *Raleway-Medium*, while *h1* and *h2* element selectors make use of *color2* variable. This means that if you change the values of *Raleway-Medium* mixin you affect *span*, *h3*, and *h4* element selectors. Furthermore, if you change the value of *color2* variable you affect the *h1* and *h2* element selectors.

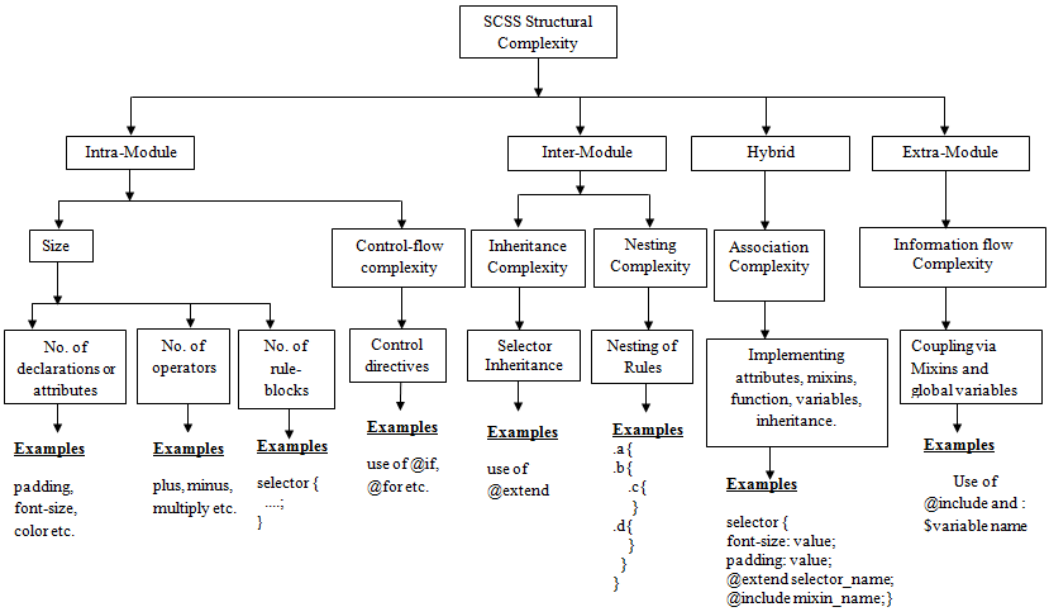


Figure. 4. Proposed Structural Complexity Attribute Classification Framework for SCSS

5. VALIDATION RESULTS

This section presents the evaluation results obtained from an expert opinion survey. An expert opinion survey technique is used to identify problems, give clarity to issues under study and evaluate products [32].

5.1. Goal of the Study

The goal of the study was to evaluate the relevance and comprehensiveness of the framework from the point of view of SCSS experts.

5.2. Context Definition

SCSS experts were invited from all over the world to participate in the online survey. The Survey Monkey platform was used to host the study questionnaires. A total of 13 experts participated in the survey and were identified through snowball sampling technique**.**

5.3. Survey Operation

The respondents were provided with the SCSS attributes classification framework, a write-up explaining how to interpret the framework and a questionnaire.

5.4. Reliability of the Research Instrument

To ensure the reliability of the instrument, pretesting was carried out and Cronbach's alpha was used as the measure of reliability. As a rule of thumb, alpha values at closer ranges to 1 are considered more internally reliable [33]. As shown in Table 1, relevance achieved a Cronbach alpha of 0.894 while comprehensiveness achieved a Cronbach alpha of 0.854. Therefore, the instrument can be considered reliable since its reliability values exceeded the prescribed threshold of 0.7 [34].

Table 1. Reliability Statistics

Scale	Cronbach's Alpha
Relevance of the Framework	0.894
Comprehensiveness of the Framework	0.854

5.5. Analysis and Interpretation

Feedback from the respondents was received and thereafter checked for completeness. All questionnaires were found to be completed satisfactorily, and therefore were accepted for data analysis.

5.5.1. Respondents Demographics

The researchers first sought to establish the characteristics of the respondents, and so characteristics such as the level of education, years of industrial experience, level of knowledge for software engineering processes and level of knowledge of SCSS was considered from all respondents.

- **Level of education for respondents**

Findings indicate that 11 (84.6%) of the respondents are bachelor's degree holders while the remaining 2 (15.4%) respondents have master's degree qualifications. These results indicate that all the SCSS experts involved in this study have attained at least the bachelor's degree, implying that they can study the framework and respond accordingly. These findings are shown in Table 2.

Table 2. Level of Education

Level of Education	Frequency	Percent (%)
Bachelors	11	84.6
Masters	2	15.4

- **Years of industrial experience**

This research sought to find the number of years the respondents have worked in the industry. It was observed that 2 of the respondents had an experience of between 2-3 amounting to 15.4% while the rest of the respondents had 4 years of experience or higher. This implies that the respondents in this study are highly experienced in the software engineering field and can be considered as experts.

Table 3. Years of Industrial Experience

Years of Industrial Experience	Frequency	Percent (%)
2-3 Years	2	15.4
4-5 Years	6	46.2
6-7 Years	2	15.4
Above 7 Years	3	23.1

- **Level of knowledge in software engineering process**

An analysis of respondent's level of knowledge was also conducted as indicated in Table 4. Findings indicate that 12 respondents representing 92.3% had high level of knowledge while 1 respondent representing 7.7% had a very high knowledge of software engineering processes. These findings imply that all participants can be trusted for analysis and opinions on the state of artefacts that are intended for use in the software engineering process.

Table 4. Level of knowledge for software engineering process

Level of Knowledge for Software Engineering Processes	Frequency	Percent (%)
High	12	92.3
Very High	1	7.7

- **Level of knowledge for SCSS**

Since the proposed framework focuses only on the structural complexity of code developed using the SCSS language, all respondents are expected to be knowledgeable SCSS programmers. Findings indicate that 8 respondents had a high level of knowledge and this corresponding to 61.5%, 3 respondents corresponding to 23.1% had moderate level of knowledge, and 2 respondents corresponding to 15.4% had a very High level of knowledge. This implies that the data collected from all the respondents can be deemed as valid. The respondents result with moderate level of knowledge are also acceptable because they can be regarded as having considerable level of SCSS knowledge in addition to their software engineering knowledge, which is acceptable for this study. These findings are shown in Table 5.

Table 5. Level of Knowledge for SCSS

Level of knowledge for SCSS	Frequency	Percent (%)
Moderate	3	23.1
High	8	61.5
Very High	2	15.4

5.5.2 Relevance of the framework

The researchers sought to know if the developed framework is relevant for the industry experts to identify the attributes that lead to SCSS complexity. Table 6. shows computed means from a Likert scale of 1 to 5 – Don't Agree, Slightly Agree, Agree, Strongly Agree and Very Strongly Agree. Findings show that the respondents agree that there is a great need for a classification framework with a mean of 3.46, which falls between agree and very strongly agree (i.e. between 3 and 4 in the Likert scale).

The respondents also agreed that the framework is useful for the process of identification of SCSS attributes as indicated by the mean of 3.62. these findings are shown in Table 6. Standard deviation was interpreted as low if the value is less than or equal to 1, while values greater than 1 are high. When the value is low it implies that the respondents didn't differ much in their opinion and high values indicate respondents considerably differed in their opinion. The standard deviation values shown in Table 6 indicates that the respondents didn't vary considerably.

Table 6. Relevance of the framework

	Need for the Framework	Usefulness of the Framework
Mean	3.46	3.62
Standard Deviation	.776	.870

5.5.3 Comprehensiveness of the framework

In a Likert scale of 1 to 5 – Don't Agree, Slightly Agree, Agree, Strongly Agree and Very Strongly Agree, respondents were asked of their opinions on whether the proposed framework is comprehensive or not. Findings show that global variables and declarations least contribute to SCSS complexity with a mean of 2.54 and 2.85 respectively. These values fall within the range of slightly agree and agree (i.e. between 2 and 3 in the Likert scale).

This implies that SCSS programmers somehow agree that the two features cause complexity in SCSS and should not be overlooked. Findings also show that all other remaining features fall in the range of agree and strongly agree (i.e. between 3 and 4 in the Likert scale). These mean values imply that the respondents agree that the concerned features contribute to SCSS complexity. The standard deviation values are high, but this is a result of the small sample size. Sullivan, [35] argued that the standard deviation of the means decreases as the sample size increases. Therefore, the high standard deviation can be explained and doesn't make the results unreliable. These results are shown in Table 7.

Table 7. Comprehensiveness of the Framework

SCSS features	**Mean**	**Standard Deviation**
Global Variables	2.54	1.127
Declaration	2.85	1.214
Operator	3.00	1.000
Control Directives	3.31	1.032
Function	3.54	1.050
Mixins	3.38	1.193
Extends	3.15	1.519
Nesting	3.46	1.561

Finally, respondents were asked whether they agree that the SCSS features identified in Table 7. wholly represents all the possible features that need to be considered when analyzing the complexity of code written in SCSS language. Findings show that 12 respondents agree corresponding to 92.3% while 1 respondent corresponding to 7.7% disagree. The findings, shown in Table 8, imply that the proposed framework is adequate as an indicator of features that cause structural complexity in SCSS code.

Table 8. Adequacy of SCSS complexity features

Adequate Features	**Frequency**	**Percent** (%)
Yes	12	92.3
No	1	7.7

6. CONCLUSION AND FUTURE WORK

In this paper, a new SCSS structural complexity attribute classification framework is proposed. The framework was validated through an expert's opinion survey. The experts agreed overwhelmingly that the framework is relevant, comprehensive and adequate, and therefore it

fully identifies the features and attributes that contribute to the structural complexity in SCSS code. This implies that the framework can be used to define structural complexity metrics for SCSS, which can then be used to show the level of complexity for SCSS code and subsequently inform the SCSS designers and programmers of the improvements that should be done on the code to improve its maintainability.

The limitation of the framework is that it's only applicable to SCSS software. Closely related CSS pre processors software's cannot use the framework to identify their structural properties. However, the framework is the first to be developed for the Cascading Style Sheets domain and therefore can be used as a guide for the development of frameworks for similar software.

The new proposed framework herein referred to as the SCSS attribute classification framework for SCSS was successfully applied to define the structural complexity metrics for SCSS [36]. However, future improvements are required to make it useful for regular CSS and CSS pre processors such as Less and stylus.

REFERENCES

[1] McCABE, T. J. (December 1976) "A Complexity Measure". IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, vol. 2, no. 4, pp. 308-320.

[2] Chidamber, S. R. & Kemerer, C. F.(1994) "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476–493.

[3] Bukhari, Z. Yahaya, J. & Deraman, A. (August 2015) "Software metric selection methods: A review". In Electrical Engineering and Informatics (ICEEI), 2015 International Conference on, IEEE, pp. 433-438.

[4] Fenton, N. & Bieman, J. (2014) "Software Metrics: A Rigorous and Practical Approach", 3rd Edition, Chapman & Hall/CRC Innovations in Software Engineering and Software Development Series.

[5] Fenton, N. & Pfleeger, S. L. (1997) "Software Metrics: A Rigorous and Practical Approach", 2nd Edition, IT Publishing Company.

[6] Morasca, S. (2015) "Rethinking Software Attribute Categorization". 6th International Workshop on Emerging Trends in Software Metrics. IEEE. pp. 31-34. DOI 10.1109/WETSoM.2015.8

[7] Daud, N. M. & Kadir, W. M. (September 2014) "Static and Dynamic Classifications for SOA Structural Attributes Metrics", Software Engineering Conference (MySEC), 2014 8th Malaysian. IEEE, pp. 130-135.

[8] Muketha, G. M. (2011) "Size and Complexity Metrics as Indicators of Maintainability of Business Process Execution Language Process Models", Doctoral dissertation.

[9] Falah, B. & Magel, K.(2015) "Taxonomy Dimensions of Complexity Metrics". Int'l Conf. Software Eng. Research and Practice.

[10] Mens, T.(2016) "Research trends in structural software complexity". arXiv preprint arXiv:1608.01533.

[11] Henderson-Sellers, B.(1996) "Object Oriented Metrics: Measures of Complexity", Prentice Hall, Upper Saddle River, NJ.

[12] Ramasubbu N. &. Kemerer, C. F. (2012) "Structural Complexity and Programmer Team Strategy: An Experimental Test", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, vol. 38, no. 5.

[13] Darcy, D. P., Slaughter, S. & Kemerer, C. F. (2005) "The structural complexity of software: An experimental test". IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, vol. 31, no. 11.

[14] Riguzzi, F. (1996) A survey of software metrics. Università degli Studi di Bologna.

[15] Morasca, S. & Briand L. C.(November 1997) "Towards a theoretical framework for measuring software attributes", In proceedings Fourth International Software Metrics Symposium, IEEE, pp. 119-126.

[16] Khan, A. A. Mahmood, A. Amralla, M. S. & Mirza, T. H.(January 2016) "Comparison of Software Complexity Metrics", International Journal of Computing and Network Technology, vol. 4, no. 1, pp. 19-26.

[17] Adewumi, A. Misra, S. & Ikhu-Omoregbe, N.(2012) "Complexity Metrics for Cascading Style Sheets". In B. Murgante (Ed.), Lecture Notes in Computer Science, vol. 7336, pp. 248-257. Springer.

[18] Misra S. & Cafer, F. (November 2012) "Estimating Quality of JavaScript" The International Arab Journal of Information Technology, vol. 9, no.6, pp. 535-543.

[19] Chung, C. & Lee, M.(1992) "Inheritance based Object-Oriented Software Metrics:, IEEE Region 10 Conference. Melbourne, Australia.

[20] Misra, S. Akman, I. &, Koyuncu, M. (June 2011) "An inheritance complexity metric for object-oriented code: A cognitive approach". Indian Academy of Sciences, vol. 36, no.3, pp. 317-337.

[21] Chawla, S. & Nath, R. (July 2013) "Evaluating Inheritance and Coupling Metrics", International Journal of Engineering Trends and Technology (IJETT), vol.4, no.7, pp. 2903-2908.

[22] Gill, N. S. & Sikka, S .(2011) "Correlating Dimensions of Inheritance Hierarchy with Complexity & Reuse". International Journal on Computer Science and Engineering (IJCSE), vol 3, no. 9, pp. 3250-3253.

[23] Li, E. Y. (1987) "A measure of program nesting complexity" National Computer Conference, San Luis Obispo, California, pp. 531-538.

[24] Chhillar, U. & Bhasin, S. (2011) "A New Weighted Composite Complexity Measure for Object-Oriented Systems". International Journal of Information and Communication Technology Research, vol. 1, no.3, pp. 101-108.

[25] Frain, B.(2013) Sass and Compass for Designers. Birmingham, UK: Packt Publishing.

[26] Stevens, W. P. Myers, G. J. & Constantine, L. L. (1974). Structured design. IBM Systems Journal, 13(2), 115-139.

[27] Li, W. & Henry, S. (1993) "Object-oriented metrics that predict maintainability" The Journal of Systems and Software, vol. 23, no. 2, pp. 111-122.

[28] e Abreu, F. B, & Melo, W. (1996), "Evaluating the impact of Object-Oriented Design on Software Quality", Proceedings of 3rd International Software Metrics Symp. Berlin.

[29] Bieman, J. M. & Ott, L. M. (1994) "Measuring functional cohesion", IEEE transactions on Software Engineering, vol. 20, no.8, pp. 644-657.

[30] Mazinanian, D. and Tsantalis, N. (March 2016) "An empirical study on the use of CSS preprocessors". In 2016 IEEE 23rd international conference on Software Analysis, Evolution, and Reengineering (SANER), pp.168-178.

[31] Cederholm, D.(2013) A BOOK APART: Sass for Web Designers. (M. Brown, E. Kissane, J. Bolton, and T. Lee, Eds.) New York, USA: Jeffrey Zeldman.

[32] Whitfield, D., Ruddock, M. & Bullman, R.(2008) "Expert opinion as a tool for quantifying bird tolerance to human disturbance". Journal of Biological Conservation, vol. 141, pp. 2708-2717.

[33] Bryaman, A., & Bell, E. (2007) Business research methods (15th ed.). Oxford: Oxford University Press.

[34] Nunnally, J. C. (1978) Psychometric theory (2nd ed.). New York: McGrawHill, 1978.

[35] Sullivan, M. (2008) Fundamentals of Statistics, Upper Saddle River, NJ: Pearson Education, Inc., pp. 382-383.

[36] Ndia, J. G., Muketha, G. M., & Omieno, K. K. (2019). Complexity Metrics for Sassy Cascading Style Sheets. Baltic Journal of Modern Computing, vol. 7, no.4, pp. 454-474. https://doi.org/10.22364/bjmc.2019.7.4.01.

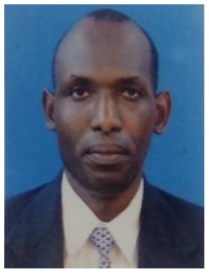
APPENDIX

```
@mixin Raleway-SemiBold {
   font-family: 'Raleway-SemiBold';
}
@mixin Raleway-Medium {
   font-family: 'Raleway-Medium';
}
@mixin PlayfairDisplay-Regular {
   font-family: 'PlayfairDisplay-Regular';
}
$color1: #f4f4f4;
$color2: #000;

p {
font-size: 5px + (6px * 2);
font-color: $color1;
@include PlayfairDisplay-Regular;
}
span{
   width: 60px;
   height: 45px;
        position: absolute;
   @include Raleway-Medium;
 }
@for $i from 1 through 4 {
.p#{$i} { padding-left : $i * 10px; }
}
@function remy ($pxsize)  {
       @return ($pxsize/16) + rem;
}

h1 {
font-size: remy(32);
font-color: $color2
}
h2{
@extend p;
font-color: $color2
```

```
}
h3 {
@include Raleway-Medium;
}
h4 {
@include Raleway-Medium;
}
h5 {
@include Raleway- SemiBold;
}

@media (min-width: 768px) {
   .modal-dialog {
      position: relative;
      top: 15%;
   }
}
```

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 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.