An Analytical Study of Cognitive Code-Level Object-Oriented Complexity Measures

Dilshan I. De Silva
Sri Lanka Institute of Information Technology
Malabe, Sri Lanka

Saluka R. Kodituwakku
University of Peradeniya
Peradeniya
Sri Lanka

Amalka J. Pinidiyaarachchi
University of Peradeniya
Peradeniya
Sri Lanka

ABSTRACT
Although several surveys on complexity measures proposed for the object-oriented approach can be found in the literature, a survey specific to cognitive code-level object-oriented complexity measures is yet to be published. Thus, a survey was conducted to make the reader aware of the cognitive code-level object-oriented complexity measures proposed since inception. Along with their calculations, the paper presents the existing cognitive code-level complexity measures in chronological order.In addition, it classifies the cognitive code-level measures based on the techniques adopted to test the validity, program component in which complexity is derived, adopted factors, the metrics used to ascertain the complexity created by each factor, capability to report program complexity as a combined value of all the considered factors, and how the program complexity value is expressed. Furthermore, the article presents the key findings uncovered from the survey and areas for development.

Keywords
Software complexity, cognitive complexity, object-oriented approach, cognitive code-level complexity measures, survey

1. INTRODUCTION
As with all the objects in the physical world, complexity is apparent in software as well. Software complexity measurement has now become an essential task that should be practiced by every organization to ensure that the developed software is ofhigh quality. IEEE defines software complexity as "the degree to which a system or component has a design or implementation that is difficult to understand and verify"[1].

The object-oriented (OO) method is believed to be more effective at regulating software complexity than the traditional procedure-oriented approach. It also supports improved quality, fast developments, cost decreases, and easier maintenance [2]. Thus, the usage of the OO approach for the development of software has become prevalent. This has led to a rise in the introduction of complexity measures for the OO approach. Based on the proposed artifact, the existing OO complexity measures can be classified into two groups: design-level and code-level measures. However, code-level measures are more effective in predicting the maintenance effort [3] and fault-prone modules [4] than the design-level measures. Furthermore, source code is regarded as the easiest artifact to compute complexity [5]. Hence, most OO measures have been proposedbased on software code.

With the introduction of cognitive informatics for the measurement of software complexity, proposing code-level OO measures based on the cognitive approach has become a popular method for measuring complexity as it computes the mental effort required to read and understand a software program.

The literature contains several surveys on existing OO complexity measures [6], [7], [8]. However, a survey specific to cognitive code-level (CCL) OO complexity measures is yet to be published. Thus, a survey was conducted to make the reader aware of the CCL OO complexity measures proposedsince inception.

Along with their calculations, the paper presents the existing CCL OO complexity measures in chronological order.The study also presents the count and type of factors considered by the studied CCL complexity measures. Furthermore, the paper categorizes the existing CCL measures based on:

- The techniquesthat were adopted to test the validity of the measures.
- The program component in which complexity is derived.
- The adopted factors
- The metrics used to ascertain the complexity created by each factor.
- The capability to report program complexity as a combined value of all theconsidered factors.
- How the program complexity value is expressed

Moreover, the article provides the following:

- The occurrence percentage of the factors used by the existing CCL complexity measures.
- The factsuncovered from the survey
- The areas for development

The rest of the paper is structured as follows. Section 2 presents the methods of the study. Section 3 provides a comprehensive description of the calculation approaches of the existing CCL OO complexity measures in chronological order. Section 4 discusses the results of the survey. Finally, Section 5 concludes the paper by summarizing the key findings uncoveredfrom the survey and suggesting areas for development.

2. METHODS
First, the existing CCL complexity measures were identified. Next, a further investigation was conducted on theuncoveredmeasures to find answers for the following:

- Who proposed the measure, and when was itproposed?
- What type of factors has the measure used, and what approaches has it used to determine the complexity introduced by each factor?
- Can the measure report the complexity of a program as a combined value of all its factors?
- What is the bottom-most level the measure can calculate values: system, class, method, variable, or program statement?
- Isthe complexity of a program expressedas an integer value?

- What techniques have been adopted to test the soundness of the measure?

The gathered data was then subjected to a thorough study to find answers for the following:

- What is the first CCL measure to be introduced?
- Have all the CCL measures been tested for their validity?
- What are the various validation techniques adopted by the CCL measures, and what CCL measure adopted the highest number of validation techniques?
- At which levels have complexity been calculated by the existing CCL measures, and what metrics have been utilized forderiving the complexity of each level?
- What CCL measures are able to report the complexity of a program as a combined value of all its factors, and at which level do they compute complexity?
- What is the total count of factors that have been taken into account by the CCL measures?
- As a percentage, how much has each factor been taken into account?
- What is the most considered factor by the studied CCL measures?
- WhichCCL measure has adopted the highest number of factors?
- What CCL measures can express program complexity as an integer value?

Finally, existing CCL measureswere classified into several categories with the intention of making the finding of the most appropriate CCL measure(s) for a given circumstance quicker and easier.

3. CCL OOCOMPLEXITY MEASURES

The first CCL measure to be proposed is the Total Complexity of OO Software Product (TCOOSP) [9]. Its value is computed as a summation of the values derived for Cognitive Complexity of Inheritance (CCI), Probability of Use of Instance Variable (PUIV), Cognitive Information Complexity of Main Function (CICMF), and Cognitive Information Complexity Metric of Classes (CICMC) metrics.

The CCI value of a program is calculatedas:

$$CCI = \left[\sum_{r=1}^{h} - \{(p/q)\log(p/q)\}_r\right] \quad (1)$$

Where:
h = Total objects in the inheritance tree
p = Count of messages sent or received by an object
q = Total messages interchanged within the inheritance tree

The PUIV value for a method is computed by dividing the instance variables used by a method from the total instance variables of the class. The PUIV value of a programis derived as the addition of total PUIV values computed for all the methods in that program.

While the CICMF value is computed based on the Cognitive Information Complexity Measure (CICM) [10] value derived for the *main* method, the summation of the CICM values calculated for all the other methods in a program provides the value of the CICMS metric.The CICM value of a method is calculated as a multiplication of the values derived for the Cognitive Weight (CW) and Weighted Information Count (WIC) metrics. The CW value is computed based on the computation technique suggested in [11]. On the other hand, the WIC value of a method*P* with *n* number of lines of code (LOC) is derived as:

$$WIC_p = \sum_{i=1}^{n} WICL_i \quad (2)$$

Where:
WICL = Weighted information count of a line of code

The WICL value is derived as:

$$WICL_k = IC_k/[n - k] \quad (3)$$

Where:
$WICL_k$=WIC value of the kth line of method P
IC_k= The total operators and identifiers in the kth line of method P

Thesecond CCL measure to be introduced is the Class Complexity (CC) measure [12]. It computes the complexity of a program (PC) as a summation of the complexity values calculated for all the methods in that program. The complexity of a method (MC) is computed based on the computation technique suggested in [11].

The third CCL measure to be introduced is the Weighted Class Complexity (WCC) measure[13]. It computes the complexity of a program (PC) as a summation of the complexity values calculated for all the classes in that program. The complexity of a class (CC) is obtained as a summation of the values derived for Attribute Complexity (AC) and Method Complexity (MC) metrics. The addition of all the attributes in a class provides values for the AC metric. On the other hand, the value of the MC metric is computed based on the computation technique suggested in [11].

The fourth CCL measure to be introduced is the metric suite suggested by Gupta and Chhabra[14]. It comprised nine metrics:Object Definition Cognitive Spatial Complexity (ODCSC), Object Member Usage Cognitive Spatial Complexity (OMUCSC), Object Member Cognitive Spatial Complexity (OMCSC), Object Cognitive Spatial Complexity (OCSC), Attribute Cognitive Spatial Complexity (ACSC), Class Attribute Cognitive Spatial Complexity (CACSC), Method Cognitive Spatial Complexity (MCSC), Class Method Cognitive Spatial Complexity (CMCSC), and Class Cognitive Spatial Complexity (CCSC).

The ODCSC value of an object *m* that is defined at line number *x* is computed as:

$$ODCSC = W_x + Distance(m, x) \quad (4)$$

Where:
Wx = Cognitive weight of the BCS that resides at line number *x*
Distance (m,x) = The absolute difference in LOC between line *x* and the line that contains the class declaration of object *m*

The OMUCSC value of an object member n that is presently used at line number x is computed as:

$$OMUCSC = W_x + Distance(n, x) \quad (5)$$

Where:
Wx = Cognitive weight of the BCS that resides at line number *x*
Distance (n,x) = The absolute difference in LOC between line x and the line in which the object member *n* has been defined in the corresponding class

The OMCSC value of an object is computed as an average of the OMUCSC values derived for all its members. The addition of the ODCSC and OMCSC values of an object provides the OCSC value of that object.

The ACSC value of an attribute *q* that is being used at line number *x* is computed as:

$$ACSC = W_x + Distance(q, x) \quad (6)$$

Where:
Wx = Cognitive weight of the BCS that resides at line number *x*
Distance (q, x) = The absolute difference in LOC between line

x and the line in which attribute *q* was previously used or defined

The CACSC value of a class is computed as an average of the ACSC values derived for all its attributes. The MCSC value of a method *j* that is presently being called or used at line number *x* is computed as:

$$MCSC = W_x + Distance(j,x) \quad (7)$$

Where:
Wx = Cognitive weight of the BCS that resides at line number *x*
Distance (j,x) = The absolute difference in LOC between line *x* and the line in which method *j* has been defined

The CMCSC value of a class is computed as an average of the MCSC values derived for all the methods of that class. The addition of the CACSC and CMCSC values provides the CCSC value of a class.

Even though the WCC [13] measure was able to compute the complexity of an OO program based on the BCSs structures of each method and the data members of each class, it was unable to capture the inheritance relationship between classes. Hence, the Cognitive Code Complexity (CCC) measure was proposed [15].The CCC measure first checks how the classes of a program are connected to capture the inheritance complexity. If there is a parent-child relationship between the classes, it computes the complexity of a program (PC) by multiplying the class complexity (CC) values derived for the parent and child classes. However, if the relationship between two classes is not a parent-child relationship, the PC value is determined as a summation of the CC values derived for those two classes.The CC value of a class is obtainedas a summation of the complexity values computed for all the methods of that class. The CCC measure uses the computation technique suggested in [11] to compute the complexity of a method.

Chhillar and Bhasin (CB) believed that software complexity is a multidimensional attribute, and hence it cannot be measured by considering a single factor [16]. With this in mind, they proposed the Weighted Composite Complexity (CB WCC) measure [16]. It computes the complexity of a program based on the values derived for the following metrics:

- Size (SZ) metric: The size of an executable statement is computed based on the number of operands, operators, methods, and strings in that statement.
- Type of control structures (TCS) metric: A weight of zero is assigned to sequential statements. Weights of 1, 2, and n are assigned to conditional control structures, iterative control structures, and a switch statement with n number of cases.
- Nesting level of control structures (NLCS) metric: A weight of zero is assigned to sequential statements. A weight of1is allocated for statements at the outermost level of nesting, 2for statements at the next inner level of nesting. Similarly, the weight allocation is increased by one for each level of nesting.
- Inheritance level (IL) metric: A weight of zero is allocated to the executable statements that reside in the base class, 1 to the executable statements at the first derived class, 2 to the statements at the next derived class. Similarly, the weight allocation is increased by one for each derived class.

The Weighted Complexity of an Executable Statement (WCES) is derived as:

$$WCES_k = SZ_k \times TW_k \quad (8)$$

Where:
$WCES_k$= Weighted complexity of the kth executable statement
SZ_k = The size of the kth executable statement
TW_k = Total weight of the kth executable statement

The summation of the values computed for the NLCS, TCS, and IL metrics provides the value of the TW metric. The value for the Total Weighted Complexity (TWC) metric is derivedas the addition of the WCES values computed for all the executable statements in a program.

The Code Complexity (CoCo) measure [17]was created to address several aspects that werenot considered by the CK metrics suite [18]. Like the CCC measure [15], the CoCo measure first checks how the classes of a program are connectedto capture the inheritance complexity. If there is a parent-child relationship between the classes, it computes the complexity of a program (PC) by multiplying the class complexity (CC) values derived for the parent and child classes. However, if the relationship between two classes is not a parent-child relationship, the PC value is determined as a summation of the CC values derived for those two classes. The CC value of a class is obtained as a summation of the values derived for Attribute Complexity (AC) and Method Complexity (MC) metrics. The addition of all the attributes in a class provides values for the AC metric. On the other hand, the value of the MC metric is computed based on the computation technique suggested in [11].

In 2012, the Cognitive Weighted Coupling Between Objects (CWCBO) measure was introduced [19]. To derive the complexity of a program (PC), it considers the occurrences and weights of five coupling types.The five coupling types and the weights allocated to them are presented in Table 1. The complexity introduced by a particular coupling type is obtainedby multiplying the weight allocated for that coupling type by the number of times it occurs in the program. Accordingly, the value of the PC metric is computed as a summation of the complexity values derived for the five coupling types.

Table 1. Weights allocated for the coupling typesconsidered under the CWCBO measure [19]

Coupling Type	Weight
Control Coupling	1
Global Data Coupling	1
Internal Data Coupling	2
Data Coupling	3
Lexical Content Coupling	4

By suggestingmodifications to the cognitive weights introduced in [11], Crasso *et al.* proposed their complexity measure in 2016 [20]. As observable from Table 2,they further divided the weight allocated fora function invocation into four types.The value of the Wm variable in Table 2 is computed based on the cognitive weight calculation approach introduced in [11]. On the other hand, based on the person who computes complexity, the weight of a *call to an abstract method* is obtained either as the summation, average, minimum, or maximum weight of the overriding methods. The complexity of a method (MC) with *k* number of linear BCS is calculated based on(9). The Weighted Class Complexity (WCC) of a class with *y* number of methods is calculated based on (11). Finally, the Code Complexity (CC) of a program with *d* levels of hierarchical depth and level *b* consisting of *c* number of classes is computed based on (13).

$$MC = \log_2(1 + [1 + MC'(MBCS)]) \quad (9)$$

Where:

$$MC'(MBCS) = \sum_{j=1}^{k} \begin{cases} W(BCS_j) * MC'\left(inner\ BCS(BCS_j)\right) \\ W(BCS_j) \end{cases} \quad (10)$$

$$WCC = \log_2(2 + WCC') \quad (11)$$

$$WCC' = AC + \sum_{i=1}^{y} MC_i^{'} \quad (12)$$

Where:
AC = Total global attributes in the class

$$CC = \log_2\left(1 + \prod_{b=1}^{d}[\sum_{h=1}^{c} WCC'_{bh}]\right) \quad (13)$$

Table 2.Weights allocated for the function invocations[20]

Type of Function Invocation	Weight
Call to a local method	2
Call to a method of an external library	3
Call to a non-local method *m*	2 + W*m*
Call to an abstract method y, for which y0, .., yn override y	I + *f*(Wy$_X$)

Byproposing enhancements to the CB WCC measure [16], the MCB measure [21] was proposed in 2018. Accordingly, in addition to taking into account the complexity introduced by the size, nesting level and type of control structures, andinheritance factors, the MCB measure takes into account the complexities introduced by the exceptions, recursion, compound conditions, pointers, dynamic memory access, references, and threads factors.The MCB measure assignsa value of 1 to the *throw* keyword under the SZ metric and a weight of 1 to the first occurrence of each *catch* statement under the TCS metricto determine the complexity introduced by the exceptions factor.To compute the recursion complexity, initially, the WCES value of a program is computedbased on (8). Then, the summation of theWCES values belonging to the executable statements of the recursive method(s) is obtained. Finally, the complexity of a program with recursive methods is calculated as a summation of the values derived for the first two steps.A weight of 1is assigned to each logical operator thatcombines two or more conditionsunder the TCS metric to determine the complexity introduced by the compound conditions.The complexity introduced by the dynamic memory access, pointers, and references factors isderivedby assigninga value of 2 under the SZ metric to *new* and *delete* reserve words and *dereference* (*) and *reference* (&) operators. Avalue of 2 is allocated to the executable statements with thread invocationsunder the SZ metric to derive the complexity introduced by threads. The value of the Total Weighted Complexity (TWC) metric is determined as an addition of the WCES values obtained for all the executable statements in a program.

4. RESULTS AND DISCUSSION

Excluding the CB WCC [16] and DSAN MCB [21] measures, all the other CCL measures compute the complexity of BCS based on the cognitive weights proposed in [11]. In fact, all those measures calculate the complexity due to BCS based on the same calculation approach suggested in [11], except for the GC [14] and CMZMP CC [20] measures. The AA CWCBO measure is the only CCL measure to be proposed based on the cognitive weight concept without considering the weights of the TCS and NLCS factors.

As depicted in Table 3, the validation approaches used by the CCL measures can be divided into three main types: theoretical, empirical, and comparative study.In addition, the following canbe observed from Table 3:

- Except for the KM TCOOSP [9] measure, all the other CCL measures have been validated at least once, either based on the theoretical or empirical approaches or via a comparative study.
- The theoretical validation of the CCL measures has been performed based on the frameworks proposed by Weyuker [22] and Briand et al. [23].
- The MA WCC [13] measure is the one to be validated with the most number of validation approaches.
- The MA WCC [13], MAK CCC [15], and AA CWCBO [19] measures are the only measures to be validated theoretically, empirically, and via a case study.
- The DSAN MCB [21] measure is the only CCL measure to be validated based on expert opinion.

Based on the program component in which the complexity is derived, the metrics of the CCL measures can be classified into several levels: application, class, method, variable, and program statement. As visible from Table 4, the lowest level that most of the CCL measures are able to compute values is the method level. However, the GC measure is even capable of calculating values at the variable level. But, the CB WCC [16] and DSAN MCB [21] measures can compute values at the program statement level. Thus, they are the only CCL measures to be blessed with this distinctive ability.

The CCL measures have revolved around twelve factors. Table 5 depicts the total number of factors and the factor types considered by each CCL measure studied for the survey. The DSAN MCB [21] measure is the one to have considered the highest number of factors. In fact, it is the only CCL measure to consider compound conditions, pointers, references, threads, and DMA factors. Table 6 provides various metrics that could be used to compute the complexities of the twelve factors.

As observable from Figure 1, with an occurrence percentage of 20%, coupling is the most considered factor by the CCL measures. Inheritance, size, TCS, and NLCS are the other factors to have obtained a percentage value of more than 10%. Thus, it is evident that most of the CCL measures have revolved around coupling, inheritance, size, TCS, and NLCS factors. Out of those five factors, inheritance is the only factor that is unique to OO programming.

Except for the GC [14] measure, all the other CCL measures are capable of providing program complexity as a combined value of all the factors considered by them.Table 7 presents the lowest level that the CCL measures compute complexity based on all their factors. As visible from Table 7, most of the CCL measures provide a single complexity value based on all their factors, only at the application level. However, there exist few measures that report a single complexity value at the class and methods levels. But, only the CB WCC [16] measure has the ability to provide a single complexity value at the program statement level based on all its factors.

As visible from Table 8, the majority of the CCL measures report program complexity as an integer value. However, few of them derive complexity as a decimal value.

Table 3. Categorization of CCL measures based on the techniques adopted to test the validity

Measure	Theoretical		Empirical		Comparative Study
	Weyuker	Briand et al	Expert Opinion	Case Study	
KM TCOOSP measure	-	-	-	-	-
MCC measure	✓	✓	-	-	-
MA WCC measure	✓	✓	-	✓	✓
GC measure	✓	✓	-	-	✓
MAK CCC measure	-	✓	-	✓	✓
CB WCC measure	-	-	-	✓	✓
MKCMZ CoCo measure	-	-	-	✓	-
AA CWCBO measure	✓	-	-	✓	✓
CMZMP CC measure	-	-	-	✓	✓
DSAN MCB measure	-	-	✓	✓	✓

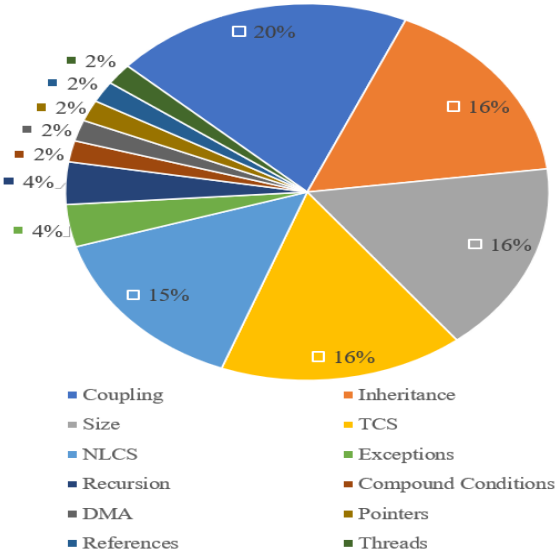

Fig. 1: Percentage values of the factors used by the CCL measures

Table 4. Categorization of CCL measures based on the program component in which complexity is derived and the metrics used to compute the complexity of each level

Measure	Application Level	Class Level	Method Level	Variable Level	Program Statement Level
KM TCOOSP measure	CCI, CICMC	-	PUIV, CICMF	-	-
MCC measure	PC	-	MC	-	-
MA WCC measure	PC	AC, CC	MC	-	-
GC measure	-	ODCSC, OMUCSC, OMCSC, OCSC, CACSC, CMCSC, CCSC	MCSC	ACSC	-
MAK CCC measure	PC	CC	-	-	-
CB WCC measure	TWC	-	-	-	SZ, NLCS, TCS, IL, WCES
MKCMZ CoCo measure	PC	AC, CC	MC	-	-
AA CWCBO measure	PC	-	-	-	-
CMZMP CC measure	CC	AC, WCC	MC	-	-
DSAN MCB measure	TWC	-	CR	-	SZ, NLCS, TCS, IL, WCES

Table 5. Factor count and factor types of CCL measures used for the survey

Measure	Factor Count	Short Name	Factor Types
KM TCOOSP measure	4	M1	Inheritance, size, TCS, NLCS
MCC measure	3	M2	Coupling, TCS, NLCS
MA WCC measure	4	M3	Coupling, size, TCS, NLCS
GC measure	2	M4	Size, TCS
MAK CCC measure	5	M5	Coupling, inheritance, TCS, NLCS, recursion
CB WCC measure	4	M6	Inheritance, size, TCS, NLCS
MKCMZ CoCo measure	5	M7	Coupling, inheritance, size, TCS, NLCS
AA CWCBO measure	1	M8	Coupling
CMZMP CC measure	6	M9	Coupling, inheritance, size, TCS, NLCS, exceptions
DSAN MCB measure	11	M10	Inheritance, size, TCS, NLCS, recursion, exceptions, threads, compound conditions, DMA, references, pointers

Table 6. Categorization of CCL measures based on the metrics used to ascertain the complexity created by each factor

Measure	Coupling	Inheritance	Size	TCS	NLCS	Exceptions	Recursion	Compound Conditions	DMA	Pointers	References	Threads
M1	-	CCI	CICMC, CICMF, PUIV	CICMC, CICMF	CICMC, CICMF	-	-	-	-	-	-	-
M2	MC, PC	-	-	MC, PC	MC, PC	-	-	-	-	-	-	-
M3	MC, CC, PC	-	AC, CC, PC	MC, CC, PC	MC, CC, PC	-	-	-	-	-	-	-
M4	-	-	ODCSC, OMUCSC, OMCSC, OCSC, ACSC CACSC, MCSC, CMCSC, CCSC	ODCSC, OMUCSC, OMCSC, OCSC, ACSC, CACSC, MCSC, MCSC, CCSC	-	-	-	-	-	-	-	-
M5	CC, PC	PC	-	CC, PC	CC, PC	-	CC, PC	-	-	-	-	-
M6	-	WCPS	WCPS	WCPS	WCPS	-	-	-	-	-	-	-
M7	MC, CC, PC	PC	AC, CC, PC	MC, CC, PC	MC, CC, PC	-	-	-	-	-	-	-
M8	PC	-	-	-	-	-	-	-	-	-	-	-
M9	MC	CCI	AC	MC	MC	MC	-	-	-	-	-	-
M10	-	WCPS	WCPS	WCPS	WCPS	WCPS	CR	WCPS	WCPS	WCPS	WCPS	WCPS

Table 7. CCL measures that report the complexity of a program as a combined value of all the considered factors and the bottom-most level that they compute complexity

Measure	The bottom-most level complexity is computed as a combined value of all the considered factors
KM TCOOSP measure	Application
MCC measure	Method
MA WCC measure	Class
MAK CCC measure	Application
CB WCC measure	Program statement
MKCMZ CoCo measure	Application
AA CWCBO measure	Application
CMZMP CC measure	Application
DSAN MCB measure	Method

Table 8. Categorization of CCL measures based on how complexity is expressed

Measures that report the complexity as a decimal value	Measures that report the complexity as a integer value
KM TCOOSP measure	MCC measure
GC measure	MA WCC measure
CMZMP CC measure	MAK CCC measure
	CB WCC measure
	MKCMZ CoCo measure
	AA CWCBO measure
	DSAN MCB measure

5. CONCLUSION

This paper provides a comprehensive overview of the CCL OO complexity measures proposed since inception.The key findingsuncovered from the survey are:

- The KM TCOOSP [9] measure is the only CCL measure not to be validated.
- Except for the CB WCC [16] and DSAN MCB [21] measures, all the other CCL measures compute the complexityofBCS based on the cognitive weights proposed in [11].
- The theoretical validation of the CCL measures has been performed based on the frameworks proposed by Weyuker [22] and Briand et al. [23].
- The CCL measure to be validated with the most number of validation approaches is the MA WCC [13] measure.
- The MA WCC [13], MAK CCC [15], and AA CWCBO [19] measures are the only measures to be validated theoretically, empirically, and via a case study.
- The DSAN MCB [21] measure is the only CCL measure to be validated based on expert opinion.
- A total of twelve factors have been adopted by the existing CCL measures to compute program complexity.
- A majority of the CCL complexity measures have revolved around coupling, inheritance, size, TBCS, and NLCS factors, with the most used factor being coupling.
- The CCL complexity measure to consider the highest number of factors is the DSAN MCB [21] measure.
- Except for the GC [14] measure, all the other CCL measures are able to provide the complexity of a program as a combined value of all the considered factors.

As an area for development, there exists a compelling need to validate the existing CCLcomplexity measures by assessing their relationship with other quality attributes.

6. REFERENCES

[1] IEEE Computer Society, *IEEE Standard for Software Quality Metrics Methodology*, IEEE Std.1061-1998, Dec. 1998.

[2] D. A. Taylor, *Object-Oriented Technology: A Manager's Guide*,2nd ed., Addison-Wesley Professional, Sep. 1997.

[3] W. Li and S. Henry, "Maintenance metrics for the object-oriented paradigm," in *Proc. First International Software Metrics Symposium*, Baltimore, MD, USA, May 1993, pp. 52-60.

[4] Y. Jiang, B. Cukic, T. Menzies, N. Bartlow, "Comparing Design and Code Metrics for Quality Prediction," in *Proc. Fourth International Workshop on Predictor Models in Software Engineering*, Leipzig, Germany, May 2008, pp. 11-18.

[5] A. Oram, G. Wilson, *Making Software: What Really Works, and Why We Believe It*, 1st ed. O'Reilly Media, Inc., USA, Oct. 2010, pp. 125.

[6] M. Sharma, N. S. Gill, and S. Sikka, "Survey of object-oriented metrics: focusing on validation and formal specification," *ACM SIGSOFT Software Engineering Notes*, vol. 37, no. 6, pp. 1-5, Nov. 2012.

[7] B. M. Goel and P. K. Bhatia, "An overview of various object oriented metrics," *International Journal of Information Technology & Systems*, vol. 2, no. 1, pp. 18-27, Jan. 2013.

[8] K. P. Srinivasan and T. Devi, "A comprehensive review and analysis on object-oriented software metrics in software measurement," *International Journal of Computer Science and Engineering*, vol. 6, no. 7, pp. 247-261, July 2014.

[9] D. S. Kushwaha and A. K. Misra, "Cognitive information complexity measure of object-oriented software - a practitioner's approach," in *Proc. Fifth WSEAS International Conference on Software Engineering*, Parallel and Distributed Systems, Madrid, Spain, Feb. 2006, pp. 174-179.

[10] D. S. Kushwaha and A. K. Misra, "A modified cognitive information complexity measure of software," *ACM SIGSOFT Software Engineering Notes*, vol. 31, no. 1, pp. 1-4, Jan. 2006.

[11] J. Shao and Y. Wang, "A new measure of software complexity based on cognitive weights," *Canadian Journal of Electrical and Computer Engineering*, vol. 28, no. 2, pp. 69-74, Apr. 2003.

[12] S. Misra, "An object oriented complexity metric based on cognitive weights," in *Proc.Sixth IEEE International Conference on Cognitive Informatics*, Aug. 2007, pp. 134-139.

[13] S. Misra and K. I. Akman, "Weighted class complexity: a measure of complexity for object oriented system," *Journal of Information Science and Engineering*, vol. 24, no. 6, pp. 1689-1708, Nov. 2008.

[14] V. Gupta and J. K. Chhabra, "Object-oriented cognitive-spatial complexity measures," *International Journal of Computer Science and Engineering*, vol. 3, no. 6, pp. 122-129, Mar. 2009.

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

[16] U. Chhillar and S. Bhasin, "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, July 2011.

[17] S. Misra, M. Koyuncu, M. Crasso, C. Mateos, and A. Zunino, "A suite of cognitive complexity metrics," in *Proc. Twelfth International Conference on Computational Science and its Applications*, Berlin, Heidelberg, June 2012, pp.234-247.

[18] S. R. Chidamber and C.F. Kemerer, "A metrics suite for object oriented design," IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476-493, June 1994.

[19] A. Aloysius and L. Arockiam, "Coupling complexity metric: a cognitive approach," *International Journal of Information Technology and Computer Science*, vol. 4, no. 9, pp. 29-35, Aug. 2012.

[20] M. Crasso, C. Mateos, A. Zunino, S. Misra, and P. Polvorín, "Assessing cognitive complexity in java-based object-oriented systems: Metrics and tool support," *Computing and Informatics*, vol. 35, no. 3, pp. 497-527, Nov. 2016.

[21] D. I. De Silva, S. R. Kodituwakku, A. J. Pinidiyaarachchi, and N. Kodagoda, "Enhancement to an OO metric: CB measure," *Journal of Software*, vol. 13, no. 1, pp. 72-81, Jan. 2018.

[22] E. J. Weyuker, "Evaluating Software Complexity Measure," *IEEE Transaction on Software Engineering*, vol. 14, no. 9, pp. 1357-1365, Sep. 1988.

[23] L .C. Briand, S. Morasca, V. R. Basili, "Property based Software Engineering Measurement," *IEEE Transactions on Software Engineering*, vol. 22, no. 1, pp. 68-86, Jan. 1996.