[[1]](#footnote-1)

**An Improvement of Cognitive Complexity Metrics**

D. I. De Silva 1, Samitha Vidhanaarachchi 2, D.K.C Senevirathna 3, Nadula J.K.D.P 4, Mapatuna R.M 5

Kulathunga H.M.C.K 6

*Abstract*--In this paper, we deep dive into cognitive complexity metrics. We have proposed some new cognitive complexity measures to evaluate several important features of object oriented (OO) languages. To be specific, we developed Recursion complexity to calculate cognitive complexity of recursive functions, Object Creation complexity to find the cognitive complexity when creating objects, and Method Chaining complexity to cover method chaining cognitive complexity. We also developed Compound Condition complexity to find how complexity changes when there are multiple condition statements, and Exception complexity to find complexity when there is nested try catch blocks. To calculate the cognitive complexity of interfaces and abstract classes, we proposed Interface complexity and Abstract Class complexity and, we developed Thread complexity to find the complexity when new threads are created. All these metrics have critically examined through theoretical and empirical validation processes.

*Keywords – object oriented, cognitive complexity, cognitive complexity metrics, cognitive informatics, cognitive code level.*

# Introduction

N

owadays software used in almost every sector in society. With the continues improvement of user requirements, businesses need to develop their software according to those requirements. Because of that sometimes complexity of software systems can go beyond control and leaving the software system to be defective. In order to provide a solution for this software engineering discipline has established some metrics to measures the complexity of a software. Through those metrics, we can easily observe what are the weaknesses of a software system and by using that we can estimate about the quality of that software. [1]

When consider about software quality, today’s business demand defect free software adherence to timeline and budget. Because their goal is to provide services to clients with minimum cost and maximized efficiency. However, over the period of time the size and complexity of software systems increased and it leads to “Software Crisis”.

Software Crisis referred to these problems given below.

* Failure at customer site
* Untraceable errors after delivery
* Difficulty in maintenance
* Time and cost slippage

So, it is must to measure the complexity of a software system and estimate the quality of that software, by using the complexity measurements.

According to the Morsani College of Medicine,

“Cognitive informatics is a new area of multidisciplinary study that involves human information processing and how those process and mechanisms relate to computing and computer applications.” [2]

In simple words cognitive weight of a software is the degree of difficulty and effort required for comprehending a given piece of software. Cognitive weight modelled by number of basic control structures in that given code or software. There are various number of ways to measure the complexity of a software. But among all these ways cognitive informatics play a major role when measuring software complexity. Because developers can calculate the cognitive weight of their software product and they can take decisions with the result. If developers get a high value for the cognitive weight, that means the complexity of that code/software is somewhat high. Then they can take necessary actions, in order to reduce the complexity. They can get an idea about whether their product (software/code) has a high readability and understandability or low.

A variety of cognitive code-level (CCL) object-oriented complexity measures have been proposed since beginning of the 1990.As examples we can get MKCMZ Coco measure [3], MOOD metrics for OO Design [4], KM TCOOSP measure [5], MCC measure [6], MA WCC measure [7], GC measure [8], MAK CCC measure [9], AA CWCBO measure [10], DSAN MCB measure [11], SALR Metrics Suite [12], SB CCC measure [13] and several other metrics [14]can be found in literature. All the given measures are tried to cover some feature (concept) of object-oriented languages. Other than that, all the measures that mentioned are developed for some quality attributes such as portability, usability, reusability interoperability, maintainability, integrity, testability, reliability and flexibility [15].

In this research we are going to propose some improvements to the MKCMZ Coco measure that developed by Sanjay Misra, Murat Koyuncu, Marco Crasso, Cristian Mateos and Alejandro Zunino who has worked on Department of Computer Engineering of Atilim University, Turkey. We are going to point out what are the features and concepts, this measure does not cover [16]. And propose some new improvements and new suite of metrics. [17]

The paper organized as follows: Next sections (section 2) is the motivation of the work section that provide information about what are the areas that COCO measure does not cover. Section 3 presents the proposal of the new suite of complexity metrics. In section 4 we demonstrate the newly proposed metrics using real-world object-oriented example and conclusion is in the final section.

# Body

## **MKCMZ CoCo Measure**

MKCMZ CoCo measure is a cognitive code level object-oriented complexity matric that proposed by group of four members. It is mainly focused on five type of factors and they are, [18],

1. Coupling
2. Inheritance
3. Size
4. Type of control structure (TCS)
5. Nesting level of control structures. (NLCS)

Based on these factors they developed some metrics such as,

* MC – Method complexity
* CC – Code complexity
* AC - Attribute complexity
* CWC- Coupling weight for class
* WCC – Weighted class complexity

Based on the factors that the focus, metrics can be divided as follows. [19]

|  |  |
| --- | --- |
| Coupling | MC, CC, CWC |
| Inheritance | CC |
| Size | AC, CC |
| Type of control structure (TCS) | MC, CC |
| Nesting level of control structures. (NLCS) | MC, CC |

They used a case study to test the validity of proposed factors. And with the all metrices they proposed we can calculate the complexity of an application easily and precisely. [20]

## **Motivation**

CoCo metric suite can be used to calculate the cognitive complexity of a given code (program) easily and precisely. Because rather that other object-oriented metric suites that available in the industry CoCo metric suite provided a set of well-defined and understandable metrices. On the other hand, these matrices do not cover the following issues. [3]

1. The impact of recursion to the cognitive complexity of a program.
2. The impact of threads to the cognitive complexity of a program.
3. The impact of Exception handling to the cognitive complexity of a program.
4. The impact of compound conditions to the cognitive complexity of a program.
5. The impact of object creation to the cognitive complexity of a program.
6. The impact of method chaining to the cognitive complexity of a program.
7. The impact of Interfaces to the cognitive complexity of a program.
8. The impact of Abstract classes to the cognitive complexity of a program.

The lack of above features, using CoCo metric suite we cannot calculate the cognitive complexity in all the instances. So, in order to make this metric suite more usable and efficient we need to improve this metric suite by adding new suite of metrics. Because lack of above-mentioned features in CoCo metric suite motivated us to produce a new suite of matrices by improving the CoCo matric suite. [3]

## **Proposed Suite of Metrics**

### **Recursion complexity (RC)**

When we take the recursion there are two types of recursive method calls. They are “local recursive calls” (LRC) and “complex recursive calls” (CRC). When proposing this metric, we only consider about the local recursive calls. LRC is the simplest recursive function and that include only one method, calling itself in a loop. When calculating the cognitive complexity of a recursive method we cannot get the exact number of iteration that the method is going to execute, so we need get the method complexity of the recursive method in each iteration with the weight of recursion method call. When the break condition occurs, it will execute only the if condition and the return statement. Accordingly, the Recursioncomplexity of a method (RC) is given by:

In this equation MC means the method complexity (proposed in CoCo metric suite) of the recursive method. Where, 2 is the weight of the method call to the recursion method (inside the recursion method) and for all iteration we get the weight of the recursion method call.” n” is the number of iterations (recursions). The outer most 2 means the weight for the break condition (if statement weight is 2).

### **Object Creation complexity (OCC)**

when considering a cognitive complexity of object-oriented based programing, object creating is a key feature. So, when we calculating complexity definitely, we need to consider the object creation complexity. When calculating the complexity of creating an object, we need to consider about some factors. They are,

* Whether there is user defined constructor in the class.
* Whether the class is a sub class.

For each instance we are going propose metrices.

1. If there is no user defined constructer in the class, OCC is given by:

In this equation, 2 is the weight of the constructor calling.

1. If there is a user defined constructer in the class, OCC is given by:

In this equation also, 2 is the weight of the constructor calling and MC is the method complexity of the constructor that is defined by the developer.

1. If the class is a sub class of an inheritance hierarchy, with the user defined constructors and call the super class constructor by ***super*** keyword. OCC is given by:

In this equation also, 2 is the weight of the constructor calling and MCi is the method complexity of the sub class constructor that is defined by the developer. Where *“li”* is the number of super class levels in the inheritance hierarchy.

### **Thread complexity (TC)**

When creating threads there are two main methods to create a thread. First method is by extending the Thread class and the second method is by implementing Runnable interface. When calculating the cognitive complexity of a program, it is must to get the complexity related to the threads. Because thread hold some considerable complexity with it. when calculating the cognitive complexity of threads, we need to consider about the object creation complexity (OCC) and the method complexity of the run methods of the threads. Accordingly, the Threadcomplexity of a method (TC) is given by:

1. Thread complexity when extending the ***Thread*** class.

In this equation OCCi is the Object creation complexity of, creating an object from the class that extends the Thread class. Where, MCi is the method complexity of the run method of that class. 2 means the complexity of executing the run method.

1. Thread complexity when implementing the ***Runnable*** interface.

In this equation OCCi is the Object creation complexity of, creating an object from the class that implements the Runnable interface. OCCTi  is the object creation complexity for, creating object from the Thread class. Where, MCi is the method complexity of the run method of that class. 2 means the complexity of executing the run method.

### **Method Chaining complexity (MCC)**

Method chaining is a method that eliminates an extra variable for each intermediate step. By using method chaining developer is saved from the cognitive burden of naming variables and keep those variable names in mind. When calculating method chaining complexity for a code segment, we need to consider about the number of chaining methods in the statement and method complexity of each method. Accordingly, the Method Chainingcomplexity of a method (MCC) is given by:

In this equation 2 means the complexity of calling first method in the method chain. Where “m” means the number of methods in the method chain. MCi is the method complexity of each method in the method chain.

### **Compound Condition complexity (CCC)**

In compound condition statements, there are multiple logical statement group together with curly braces. When we are going to calculate the cognitive complexity of compound conditions there are some factors that we need to consider. First, we need to consider about the type of logical operations that used in the compound conditions and also, we need to consider about the linear flow of the compound condition. By considering these factors we proposed metrics for each instance of compound conditions.

1. Compound condition that has only one logical operator (AND OR). Then compound condition complexity is given by:

In this equation 2 is the complexity for the if statement. If there is only one logical operator used in the compound condition, the linear flow will cautiously remain. So, in that case we can get the complexity as 2.

1. Compound condition that has multiple logical operators. Then compound condition complexity is can be computed by considering following factor.

* Every time when linear flow breaks it add 1 complexity.

Let’s consider the following if statements with different sequence of logical operators.

* If (AND AND OR OR)

In this case the linear flow of the compound condition is continuously remain until it meets the OR operator. So, after it met linear flow will break and 1 complexity need to added. Hence total complexity is 2 + 1 =3.

* If (AND OR AND)

In this case the linear flow only remains until it meets the first OR operation, after first OR met it breaks the linear flow and add 1 complexity total. After that linear flow of OR operations, will remains until it meets the second AND.second AND breaks the flow again and add 1 complexity to the total. Hence the total complexity become 2 + 1 + 1 = 4.

* If (OR OR AND AND)

In this case the initial linear flow remains until it meets first AND operator. After it met the first AND operator the flow will break and add 1 complexity to the total. After that there are no any different operation type to break the flow. Hence the total complexity become 2 + 1 = 3.

### **Exception complexity (EC)**

When we handle, the exceptions using try catch blocks in Java applications, there can be so many try catch blocks in our code. So, in order to precisely compute the cognitive complexity, we need to consider those, try catch blocks also. So, when we calculating the complexity of try catch blocks we need to consider about the nested level among those blocks. We proposed that each try catch block will contain 1 complexity and if that try catch block is in nested level, complexity is the number of the nested level that specific try catch block contains. Accordingly, the Exception complexity of a method (EC) is given by:

In this equation,

*m* = Number of nested levels

*l* = Nested level number (current)

*Nl =* Number of try catch blocks in *l* level

### **Interface complexity (IC)**

In CoCo metric suite they already consider about the complexity of a class by proposing weighted class complexity (WCC) metric. But they did not consider about the interfaces. So, we proposed a metric for computing complexity of an interface. When proposing the metric, we consider about the number of attributes in the interface and number of methods (abstract methods) in the interface. Accordingly, the Interface complexity of a method (IC) is given by:

In this equation AC (attribute complexity proposed in CoCo) means the complexity of the attributes that contains in the interface. Where 2 means the complexity of each method(abstract) in the interface. “n” means the number of methods in the interface.

### **Abstract Class complexity (ACC)**

CoCo metric suite also did not focus on the complexity of abstract classes. So, we proposed a metric to compute the complexity of an abstract class. When proposing the metric we consider about the number of attributes, number of abstract classes, and number of normal classes in the abstract class. Accordingly, the Abstract class complexity of a method (ACC) is given by:

In this equation AC (attribute complexity proposed in CoCo) means the complexity of the attributes that contains in the abstract class. Where 2 means the complexity of each abstract method in the interface. “n” means the number of methods in the interface. Where “p” means number of normal methods in abstract class and MCp means the method complexity of each normal method.

### **Average Recursion complexity (ARC)**

This gives the average recursion complexity of a code or program. We developed this metric by adding all Recursion complexity (RC) for all recursive functions in the program and dividing it by number of recursive functions in program.

Where RC is recursion complexity of a particular recursive method, m is total number of recursive methods in the program.

### Average Object Creation complexity (AOCC)

This gives the average object creation complexity of a program. We developed this metric by adding all object creation complexity (OCC) for all object creational statement in the program and dividing it by number of object creational statements in the program

Where OCC is object creation complexity of a particular object creational statement, m is total number of object creational statement in the program.

### **Average Thread complexity (ATC)**

This gives the average Threadcomplexity of a code or program. We developed this metric by adding all Threadcomplexity (TC) for all threads in the program and dividing it by number of threadsin the program

Where TC is Threadcomplexity of a particular thread, m is total number of threads in the program.

### **Average Method Chaining complexity (AMCC)**

This gives the average Method chainingcomplexity of a program. We developed this metric by adding all Method chainingcomplexity (MCC) for all Method chaining statements in the program and dividing it by number of Method chaining statements in the program

Where MCC is Method chainingcomplexity of a particular Method chaining statement, n is total number of Method chaining statements in the program.

### **Average Compound Condition complexity (ACCC)**

This gives the average Compound conditioncomplexity of a program. We developed this metric by adding all Compound conditioncomplexity (CCC) for all Compound conditional statements in the program and dividing it by number of Compound conditional statements in the program.

Where CCC is Compound condition complexity of a particular Compound conditional statement, n is total number of Compound conditional statements in the program.

### **Average Exception complexity (AEC)**

This gives the average Exception complexity of a program. We developed this metric by adding all Exception complexity (EC) for all try catch blocks in the program and dividing it by number of try catch blocks in the program

Where EC is Exception complexity of a particular try catch block, n is total number of try catch blocks in the program.

### **Average Interface complexity (AIC)**

This gives the average Interface complexity of a program. We developed this metric by adding all Interface complexity (IC) for all interfaces in the program and dividing it by number of interfaces in the program

Where IC is Interface complexity of a particular interface, n is total number of interfaces in the program.

### **Average Abstract Class complexity (AACC)**

This gives the average Abstract class complexity of a program. We developed this metric by adding all Abstract class complexity (ACC) for all abstract classes in the program and dividing it by number of abstract classes in the program

Where ACC is Abstract class complexity of a particular abstract class, n is total number of abstract classes in the program.

## **Demonstration of the Metrics**

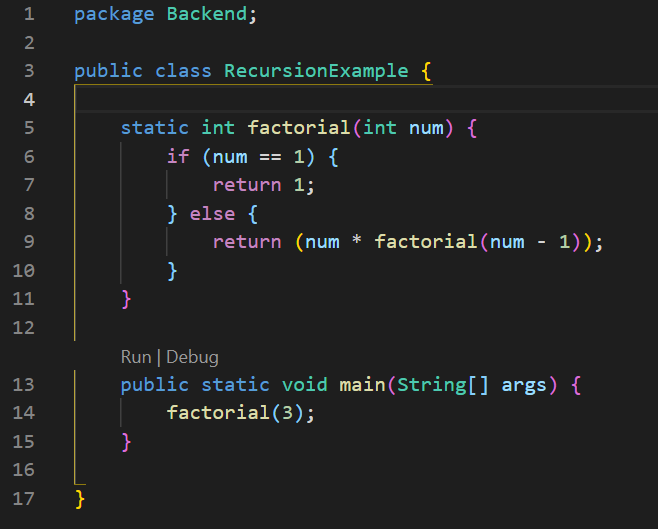


Figure : Recursion function

### **Recursion complexity (RC)**

In this given example there is a one class called RecursionExample that contains a recursive method called factorial. Now let’s look at how we can calculate the cognitive complexity of this code segment using the metric that we proposed.

* RCfactorial = 2 + MCfactorial +
* MCfactorial = 3
* n = 3
* RCfactorial = 2 + 3 + 3 = 8

### **Object Creation complexity (OCC)**

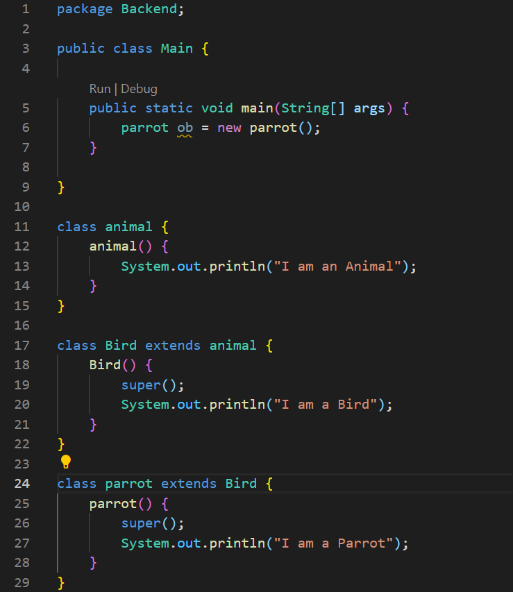


Figure : Object creation complexity

In this given example there are classes named animal, Bird, parrot that related each other with inheritance.in each sub class constructor there is super method call to execute the parent constructor. Now let’s look at how we can calculate the cognitive complexity of this code segment using the metric that we proposed.

* OCCob = 2 + *l* (MC)
* *l* = 2 (number of super class levels)
* MC = 1 (complexity of constructor of the class that used to create object)
* OCCob = 2 + 2 (1) = 4

### **Thread complexity (TC) (by extending Thread class)**

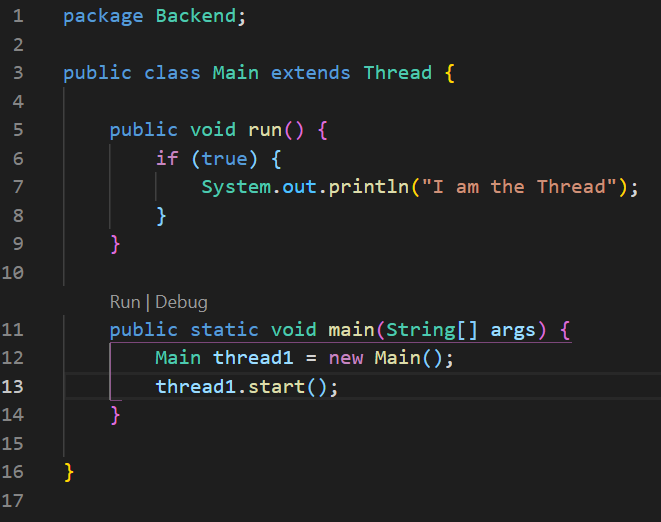


Figure :Thread creation using Thread class

In this given example there have a Main class that extends the Thread class and it defined the run method of the thread. In the main class it creates a new thread and then it starts the thread by executing start method. Now let’s look at how we can calculate the cognitive complexity of this code segment using the metric that we proposed

* TCthread1 = OCC + MC + 2
* We can assume OOCthread1 = 2
* MCrun = 1 + 2 = 3
* TCthread1 = 2 + 3 + 2 = 7

### **Thread complexity (TC) (by implementing Runnable interface)**

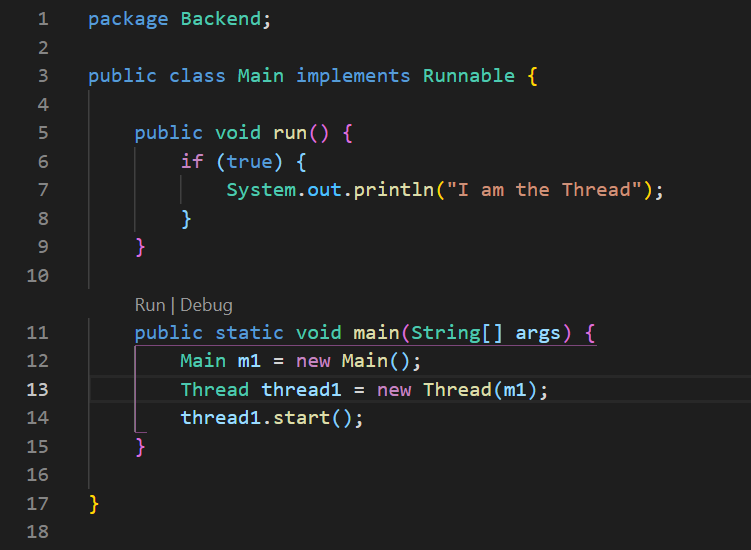


Figure : Thread creation using Runnable interface

In this given example there have a Main class that implements the Runnable interface and it defined the run method of the thread. In the main class it creates a new thread and then it starts the thread by executing start method. Now let’s look at how we can calculate the cognitive complexity of this code segment using the metric that we proposed.

* TCthread1 = OOCm1 + OOCthread1 + MCrun + 2

Figure :Compound condition

* OOCm1 = 2
* OOCthread1 we can assume as 3
* MCrun = 3
* TCthread1 = 2 + 3 + 3 + 2 = 10

### **Method Chaining complexity (MCC)**

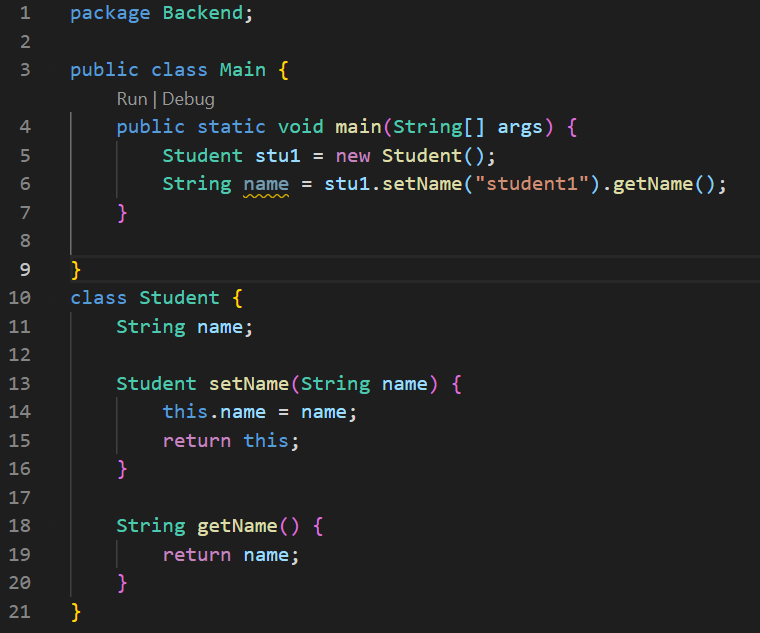


Figure :Method chaining

This example represents a method chaining code segment. Now let’s look at how we can calculate the cognitive complexity of this code segment using the metric that we proposed

* MCC = 2 + MCsetName + MCgetName
* MCsetName  = 1
* MCgetName = 1
* MCC = 2 + 1 + 1 = 4

### **Compound Condition complexity (CCC)**

This example represents a compound condition code segment. Now let’s look at how we can calculate the cognitive complexity of this code segment using the metric that we proposed

CCC = complexity of if + 1 (because of || (OR) breaks the linear flow) + 1 (because second && breaks the linear flow)

CCC = 2 + 1 + 1 = 4.

### **Exception complexity (EC)**

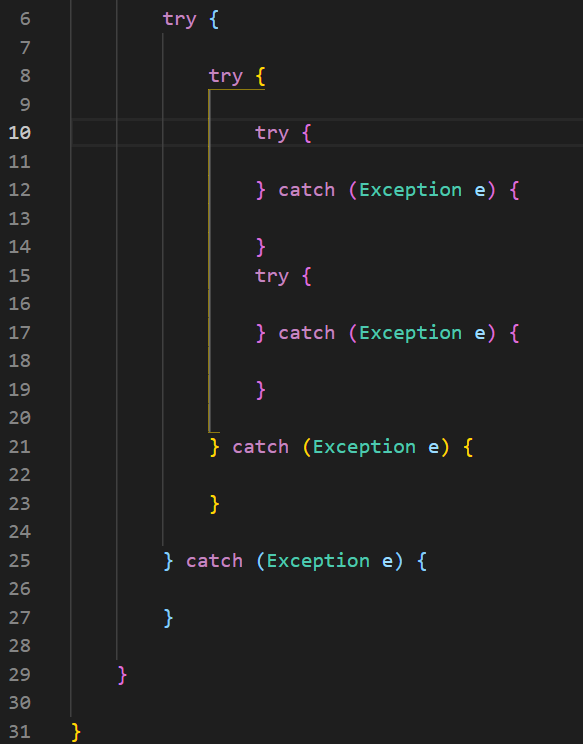


Figure 7: try catch blocks

This example represents a try catch block code segment. Now let’s look at how we can calculate the cognitive complexity of this code segment using the metric that we proposed

* Number of nested levels = 3
* EC = (1 \* 1) + (2 \* 1) + (3 \* 2)
* EC = 1 + 2 + 6
* EC = 9

### **Interface complexity (IC)**

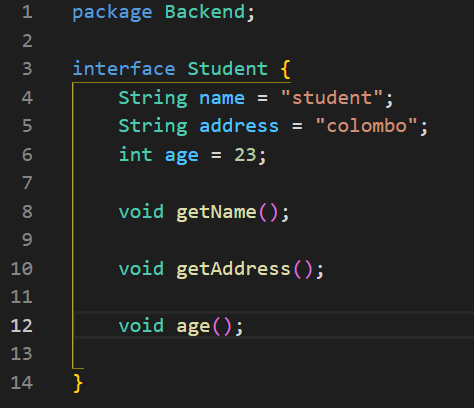


Figure 8: Interface

This example code contains a sample code of interface named student. Now let’s look at how we can calculate the cognitive complexity of this interface using the metric that we proposed

* ICstudent = ACstudent +
* ACstudent = 3
* ICstudent  = 3 + (2 + 2 + 2)
* ICstudent = 3 + 6 = 9

### **Abstract Class complexity (ACC)**

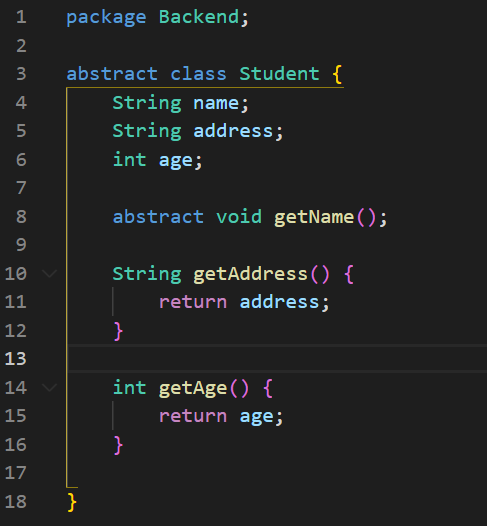


Figure : Abstract class

This example code contains a sample code of abstract class named student. Now let’s look at how we can calculate the cognitive complexity of this abstract class using the metric that we proposed

* n (number of abstract methods) = 1
* m (number of normal methods) = 2
* ACCstudent = ACstudent +
* ACCstudent = 3 + (2) + (1 + 1)
* ACCstudent = 3 + 2 + 2 = 7

# Conclusion

We are proposed based on this research paper the new cognitive complexity metrics for Objected Oriented. Because produce a high-quality development process. Most of the elements found in object-oriented programming, such as method, attribute, class, inheritance, and coupling, may be captured using these metrics. Furthermore, the objective of developing such a metric suite is to aggregate the majority of the features that cause complexity. The code complexity measures the structural and cognitive difficulty of an OO system, and these metrics calculate the complexity at each level of the code. In addition, the proposed metrics suite will be investigated in addition to improving the remaining characteristics of object-oriented code. In addition, an automated method for calculating the metrics in the suite will be built.

# REFERENCES

|  |  |
| --- | --- |
| [1] | 01 04 2022. [Online]. Available: https://www.castsoftware.com/glossary/software-complexity. |
| [2] | 5 04 2022. [Online]. Available: https://www.usfhealthonline.com/resources/health-informatics/what-is-cognitive-informatics. |
| [3] | Sanjay Misra ,Cristian Mateos ,Murat Koyuncu ,Alejandro Zunino , " A Suite of Cognitive Complexity Metrics," *Lecture Notes in Computer Science ·,* June 2012 . |
| [4] | R. C. S. a. N. R. t. . Harrison, "An Evaluation of the MOOD Set of Objec". |
| [5] | A. DHARMENDER SINGH KUSHWAHA, "Cognitive Information Complexity Measure of Object-Oriented Software – A Practitioner’s Approach," *Conf. on Software Engineering,* 2016. |
| [6] | S. Misra, "An Object Oriented Complexity Metric Based on Cognitive Weights," 2007. |
| [7] | K. I. A. SANJAY MISRA, "Weighted Class Complexity: A Measure of," *JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 24, 1689-1708 (2008),* 2008. |
| [8] | V. G. a. J. K. Chhabra, "Object-Oriented Cognitive-Spatial Complexity," *International Journal of Computer and Information Engineering,* 2009. |
| [9] | M. K. Sanjay Misra, "An inheritance complexity metric for object-oriented code: A cognitive approach," *researchgate.net,* July 2011 . |
| [10] | L. A. A. Aloysius, "Coupling Complexity Metric: A Cognitive," *I.J. Information Technology and Computer Science, 2012, 9, 29-35,* 2012. |
| [11] | S. R. K. J. P. K. D. I. De Silva, "Enhancements to an OO Metric: CB Measure," *Journal of Software,* 2017/20. |
| [12] | A. A. F.-S. D. SANJAY MISRA, "A Suite of Object Oriented Cognitive," *IEEE access,* 2018. |
| [13] | ,. P. K. B. Kulwinder Singh, "An Approach to Measure Cognitive Complexity for Object Oriented Code," *International Journal for Research in Applied Science & Engineering,* 2015. |
| [14] | C. 1. Stephen H. Kan: Metrics and Lessons Learned for OO Projects, " Metrics and Lessons Learned for OO Project," *Metrics and Models in Software Quality Engineering. Addison-Wesley (2003),* 2003. |
| [15] | J. a. D. C. Babsiya, "A Hierarchical Model for Object Oriented Design Quality assessment," *IEEE Transactions on Software Engineering, 28(1), 4-17 (2002),* 2002. |
| [16] | W. J. Briand L., "Modeling Development Effort in Object Oriented System Using Design Properties," *IEEE Transactions on Software Engineering, 27(11), 963-986 (2001),* 2001. |
| [17] | K. S. a. W. Kim, " Complexity Measures for Object-Oriented Program Based on the Entropy.," *In: Proc. Asia Pacific Software Engineering , pp. 127–136 (1995),* 1995. |
| [18] | [Online]. Available: https://www.researchgate.net/figure/An-example-for-cognitive-complexity-calculations\_fig1\_228757079. |
| [19] | [Online]. Available: https://dev.to/promyze/cyclomatic-complexity-and-cognitive-complexity-4c03. |
| [20] | [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1409/1409.4919.pdf. |

1. [↑](#footnote-ref-1)