A New Class Coupling Metric Based on Method Inheritance

Aisha Alhaitom[1], Asmaa Abdulhafid Shuwehdi[2]

[1] Monitoring Education, Al-hraifa.Ibrik Secoundary School,
aishaalsokani6030@gmail.com

[2]Research funding and support department,Libyan Authority for Scientific Research, Tripoli, Libya
a.shuwehdi@out.edu.ly

Abstract:

Object-Oriented Programming (OOP) is becoming more popular in the software development environment, and object-oriented programming standards are an essential part of the software environment. This paper focuses on a set of these standards that can be used to measure the quality of design in OOP and that are applied to a set of design classes and properties. These standards enable designers to access software early in the process, which reduces the degree of sophistication and improve design.

The paper also, presents and discusses a number of classification criteria that may serve as the evaluation system for the scale of coupling in object- oriented programming. Based on those criteria, a new object-level programming correlation scale and its verification is proposed, as well as a tool for measuring the correlation of classes. The programming language used is VB.NET.

Keywords: Class, Coupling, Cohesion, Inheritance, Metrics, OOP, Measurement.

1. INTRODUCTION:

Anything that cannot be measured, it cannot be controlled. Measurement and metrics are in practice throughout the history of mankind, they played an effective role in all vital human activities. No productivity process can be controlled and no promotion or development may be beneficial in the absence of effective metrics. In regard to software engineering, measurement and metrics help to guarantee the production of high-quality products in terms of reliability, cost effectiveness and high productivity[1].

Software engineering is considered as one of the computer science branches. It is emerged as a result of many problems and failures in software industry, or as a high cost of software constructing it. Software industry is accomplished in an engineering fashion. During its construction, it passes through many descriptive models that describe its forms, building elements, and behavior expected in its actual environment.

System software is built according to one of the following programming models:

i. Sequential Programming model.

ii. Procedural Programming model.

iii. Parallel (Concurrent) Programming model.

iv. Object-Oriented Programming Model (OOP).

Object-oriented programming has shown its popularity in the field of software development, accompanied by the emergence of several measures which are used to improve software product based on a set of characteristics such as Complexity, Size, Coupling, Cohesion, etc., and therefore, these standards provide us with a clear idea of software product quality.

Object-oriented programming has many features such as Inheritance, where a class (subclass) can inherit some or all of the attributes and/or the methods of other class (superclass). The concept of inheritance provides us reusability property, which saves a lot of effort and time during the software development property. So, it's still there are several things we can add to the inheritance metrics to develop and improve[2].

2. OBJECT-ORIENTED PROGRAMMING: APPROACH

We can understand object-oriented programming model looking and reflecting the world around us, we may see some machines such as electrical sweeper, coffee machine, and other objects everywhere. Some of these objects such as imaging machine operates independently, some of which interact with each other, such as phones and call recording, and some of the objects within the last objects such as ice inside the Ice Maker refrigerator maker. Object-oriented programming (OOP): is a programming paradigm based on the concept of "Objects", which can contain attributes and methods.

From the above, we find that object-oriented programming model depends on style objects which are identical to the objects in the real world, such as car, book, house, etc.

Where the program is divided into a set of objects, each object has Attributes, and Methods, each object followed by a specific class.

. A major advantage of OOP is code reusability. Some other important features of Object-Oriented Programming are as follows [2]

a. Emphasis on data rather than procedure.
b. Programs are divided into Objects (as mention above).
c. Data are hidden and cannot be accessed by external functions.
d. Objects can communicate with each other though functions.

3. CONCEPTS OF OOP:

OOP is characterized by a set of object-oriented programming characteristics that made it the most appropriate model to avoid problems resulting from the increased size and complexity of the software model. These characteristics as Deborah J. Armstrong [3]identified, and others are: **A. Objects and Classes:**

• An object is defined as a software part of attributes and related methods. Or, as an entity which has **attributes** (whose representation is hidden) and a defined set of **operations** which operate on that state follows [2].

One can define an object-orienetet system as an organized collection of cooperative objects representing real world entities.

The notion used to represent an object is shown in Figure (1).

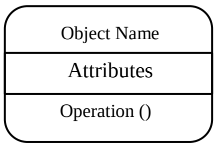


Figure (1): Notion for an Object.

Attributes are represented as a set of objects. Attributes describe and define an object by clarifying what is meant by the object in the context of the problem scope, and the operations contain control and procedural constructs that may be invoked by a message- a request to the object to perform one of its operations. An operation changes an object in some way; specifically, it changes one or more attribute values that are contained within the object.

Figure (2) shows an example of an object. The name of the Object is Chair, it has the attributes of cost, dimensions, weight, location and color, and the operations are: buy, sell and weight.

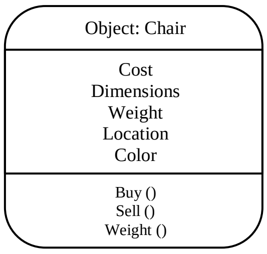


Figure (2): Example of an Object

A class defines the attributes and the methods common to all objects of a certain kind. A class is also defined as a set of objects that share a common structure and common behavior manifested by a set of methods; the set serves as a template from which objects can be instantiated (created).

Figure (3) shows Chair as a member (the term instance is commonly used) of much larger class of objects called Furniture. A set of generic attributes can be associated with every object in the class of Furniture. Because Chair is a member of the class of Furniture, the Chair inherits all attributes defined for the class. Once the class has been defined, the attributes can be reused when new instances of the class are created.

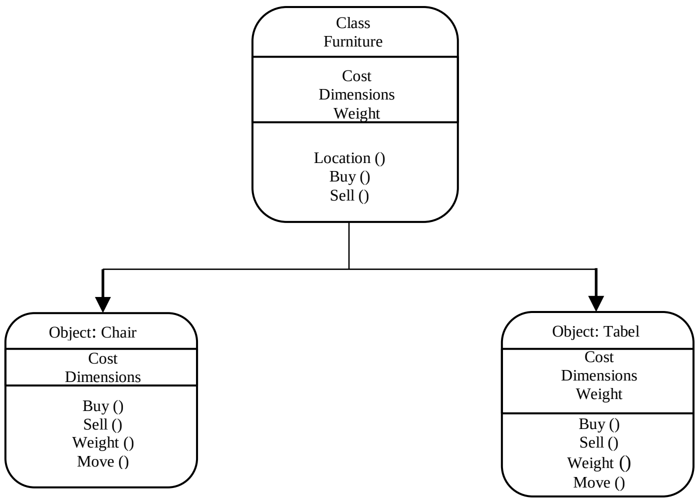


Figure (3): Class Furniture with Object and Inherited Methods

A method is an operation or set of procedural statements upon an object for achieving the desired result. It performs different kinds of operations on different data types. Software objects interact and communicate with each other by sending messages to each other. When object A wants object B to perform one of B's methods, object A sends a message to B as shown below in Figure (4).

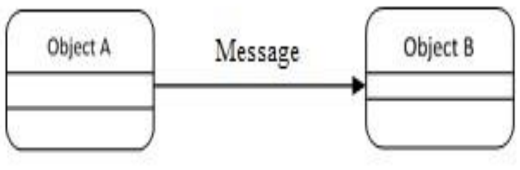


Figure (4): Message Passing B.

Encapsulation:

In object-oriented programming, encapsulation refers to one of two related but distinct notions, and sometimes to the combination thereof:

- A language mechanism for restricting direct access to some of the object's components.
- A language construct that facilitates the bundling of data with the methods (or, the functions) operating on that detection: Is the hiding of some details of a process, and

showing only the characteristics and operations required for users in order to display more clearly. C. **Abstraction**:

Is the hiding of some details of a process, and showing only the characteristics and operations required for users, in order to display more clearly.

D. **Polymorphism:**

It means different objects may respond to the same message in different ways. E. **Inheritance:**

Inheritance is a way to make new classes using predefined classes. Where a subclass inherits the attributes and methods of the superclass without the original object is affected. Figure (5) depicts Inheritance Network. For example, the HARDWARE class is a subclass of the COMPUTER class, so it inherits the attributes and methods of the its parent.

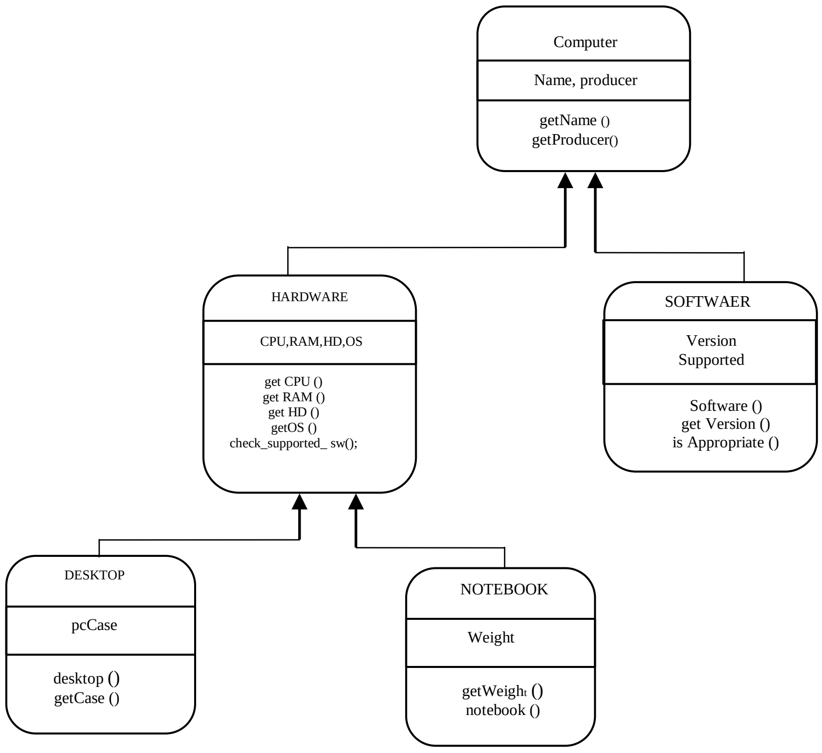


Figure (5) Inheritance Network

4. COHESION AND COUPLING:

In order to ensure that the software product is good, it must satisfy the **quality properties**, which means providing quality factors for a specific product or service, and its compliance to the required specifications such as Efficiency,

Flexibility, Integrity, Maintainability, and as well as Interoperability, that facilitates the Reusability property. Those factors are applicable to part or all the system, and to ensure the provision of quality factors mentioned earlier, there are a range of measures that should be made to ensure that the specifications of Software Quality Assurance (SQA) are applied, including:

• Cohesion:

The degree to which the methods within a class are related to one another and work together to provide well-bounded behavior. Effective object-oriented designs maximize cohesion since it promotes encapsulation. Cohesion is an internal software attribute that describes how well connected the components of a software module **[4]**

Coupling:

Coupling is a measure of the strength of association established by a connection from one entity to another. Classes (objects) are coupled in the following ways:

When a message is passed between objects.

i. When methods declared in one class use methods or attributes of the other classes.
ii. When inheritance introduces significant tight coupling between super classes and their subclasses.

Coupling and cohesion are highly related, bad cohesion usually leads to bad coupling, because they have a highly interdependent influence. Software engineering experts assure that designs with low coupling and high cohesion lead to products that are both more reliable and maintainable.

A class with low coupling is not dependent on too many other classes. Such classes may be undesirable due to the following [5]:

- Changes in related classes force local changes.
- Harder to understand in isolation.

- Harder to reuse because its use requires the additional presence of the classes on which it is dependent.

The extreme case of Low Coupling is when there is no coupling between classes. This is not desirable because a central metaphor of object technology is a system of connected objects that communicate via messages. If Low Coupling is taken to excess, it yields a poor design because it leads to a few incohesive, bloated, and complex active objects that do all the work [5].

A class with low cohesion does many unrelated things, or does too much work. Such classes are undesirable; they suffer from the following problems:

- Hard to comprehend.
- Hard to reuse.
- Hard to maintain.
- Delicate; constantly effected by change.

Figure (6), depicts an object-oriented system that starts by defining a class (Class A) containing related or similar attributes and operations (some operations are methods) [5].

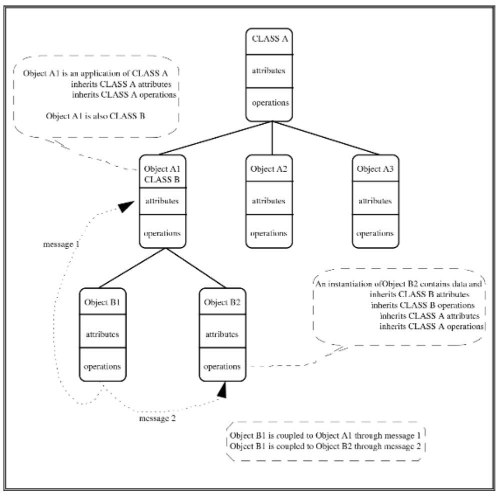

CLASS A
attributes
operations
Object A1 is an application of CLASS A
inherits CLASS A attributes
inherits CLASS A operations
Object A1 is also CLASS B
Object A1 CLASS B
attributes
operations
Object A2
attributes
operations
Object A3
attributes
operations
message 1
Object B1
attributes
operations
Object B2
attributes
operations
An instantiation ofObject B2 contains data and
inherits CLASS B attributes
inherits CLASS B operations
inherits CLASS A attributes
inherits CLASS A operations
message 2
Object B1 is coupled to Object A1 through message 1
Object B1 is coupled to Object B2 through message 2

Figure (6) Pictorial Description of Object-Oriented Terms

The classes are used as the basic for objects forming hierarchical trees. A child class inherits all of the attributes and operations from its parent class, in addition to having its own attributes and operations. A child class can also become a class for other classes, forming another branch in the hierarchical tree structure [6].

5. MEASUREMENT AND METRIC:

The use of measurement is common in everyday life. For it is used to do such things as change motor oil of a car, or when we check the time of day, or the distance we have traveled in our car. So, measurements are used extensively in most areas of production and manufacturing to estimate costs, calibrate equipment, assess quality, and monitor inventories. Measurement is the process of empirical and objective assignment of numbers to attributes of objects or entities of the real world in such a way to describe them[7]. One can classify measurement as:

1. Factors that can directly be measured (internal attributes such as cost, speed, memory, etc.…)
2. Factors that can be measured only indirectly (external attributes such as functionality, quality, complexity, and maintainability).

Measurement has two broad uses: assessment and prediction[7].

1. Assessment measures can be used for:
   - Monitoring the advancement of a project in order to take the appropriate corrective decisions.
   - Evaluating a software product or process.

   Predictive measures can be used for:
   - Planning the resources and time needed for a certain project.
   - Predicting the outcome of a project, in terms of size, quality or other attributes of the delivered software.

A metric is defined as a process by which numbers are assigned to attributes of entities in the real world to describe them according to clearly defined rules.

6. PAPER PROBLEM:

Inheritance is a key concept of object-oriented programming, where they lead to split and re-use program code by establishing a connection between methods and attributes. Other advantages (mentioned earlier) are software reusability, creating new classes from existing class and subclasses extending superclass. Because of this importance, and although there are many metrics competent measuring the degree of class coupling, there is still more to be added to it to improve and develop these metrics as reported in previous studies. In this study will be the development depending on inheritance characteristic of object-oriented programming. From that, some advantages of inheritance are:

1. Software reusability.
2. Create new class from existing class.
3. Subclass extends superclass.

7. PAPER OBJECTIVE:

The aim of the study is to propose a new object-oriented programming coupling scale for measuring class coupling depending on the inheritance property of object-oriented programming. A new approach for class coupling based on inheritance in methods would be proposed as an enhancement to the related existing metrics. The proposed metric measures the degree of class coupling based on inheritance property, it shows the possibility of using part of the software or not from the beginning of developing software.

We can summarize the methodologies of this research as shown Figure (7).

Many metrics have been proposed to estimate different attributes such as cohesion, coupling, and complexity[4].

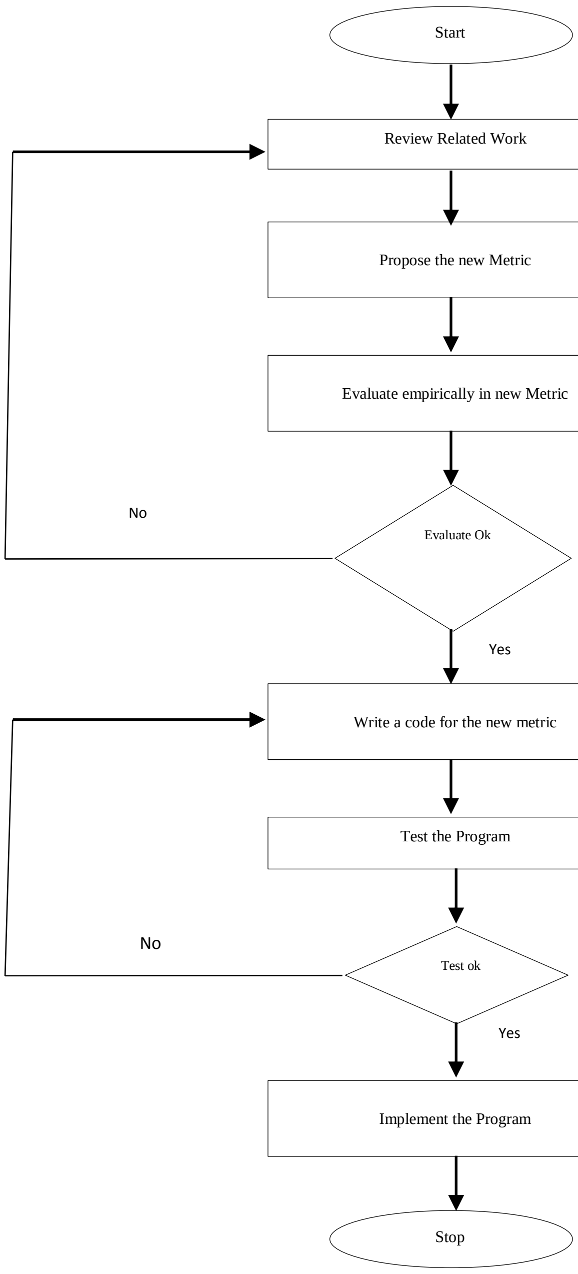


Figure (7) A methodology of the research

8. RELATED STUDIES:

8.1Classification of Object-Oriented Metrics

The development or construction of any software project goes through the collection of phases. The first of these phases is the analysis of the system, in which a summary description of the purpose of this system is developed by defining the requirements.

The requirements are defined as the services expected to be provided by the system (functional requirements or Internal attributes) and the constraints to be the system must subject to it, such as, the external form of the interfaces, reliability, performance requirements, etc. (non-functional requirements or External attributes) [4].

8.2 Categories of Object-Oriented (OO) Metrics

There are different Categories for OO metrics. But , one can classify OO metrics based on internal and external quality attributes for software system, shown in Figure (8) as mentioned in reference[4].

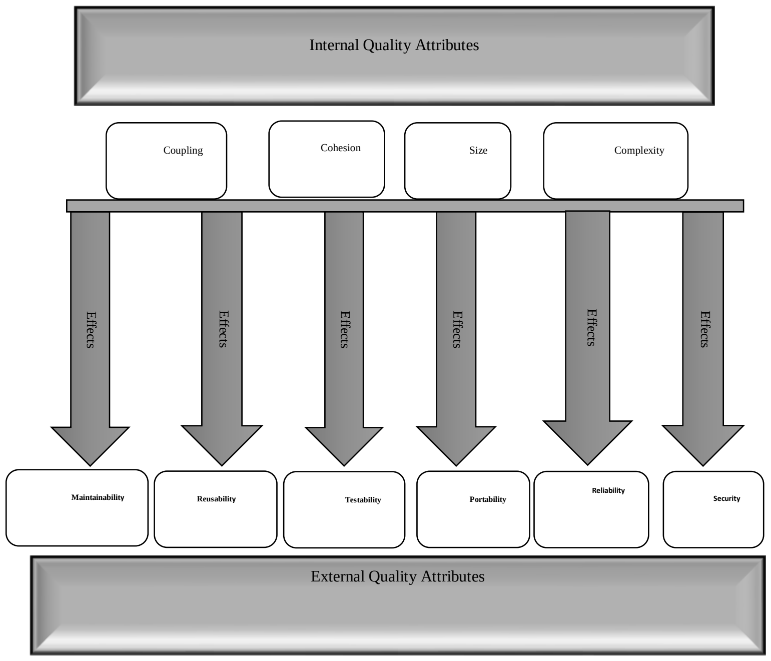


Figure (8) Categories of Object-Oriented (OO) Metrics

All internal attributes, however varied and different, are included in one of the following quality characteristics:

1. Coupling.
2. Cohesion.
3. Size.
4. Complexity.

For example, the metric proposed in this paper falls under the coupling property, and so on to the other characteristics and measurements. And every attribute has effects directly at the one external attribute or more, as in the Figure(8).

8. METHODS INVOCATION CRITERION:

Methods are like procedures that provide the interface between the object and the program using the object. Each method has an explicit return type, or specifies the return type void.

We consider the example given in Figure (9).

Method M_1 is calling method M_2 directly. Consequently, method M_1 is calling M_3 indirectly. Method M_1 shares the attribute a_1 directly and M_1 is calling M_2 directly, so method M_1 shares the attribute a_2 indirectly[8].

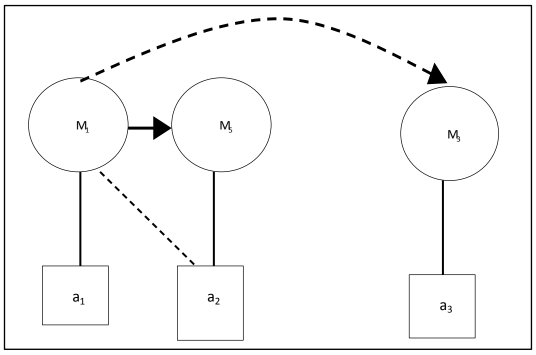


Figure (9) Method Connection

9. INHERITANCE:

Inheritance is a mechanism in which one object acquires characteristics from one, or more other objects, and it is the sharing of attributes and methods among classes based on a hierarchical relationship[4].

Inheritance occurs at all levels of a class hierarchy. It is used to construct relationship between super classes and subclasses in various ways. Types and metrics of inheritance are discussed below:

A. Types of Inheritance

i. **Single Inheritance:** is a common form of inheritance in which classes have only one base class as shown in Figure (10).

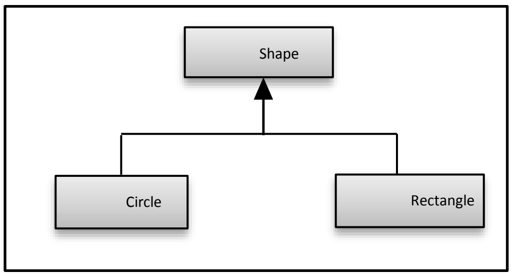


Figure (10) Single Inheritance

ii. **Hierarchical (Repeated) Inheritance:** is the hierarchy in which multiple subclasses inherit from one base class. as shown in Figure (11).

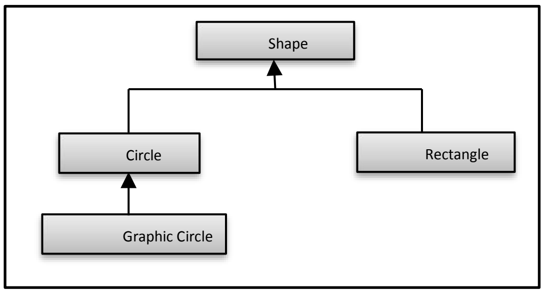


Figure (11) Repeated Inheritance

iii. **Multiple Inheritance** as shown in Figure (12), is the hierarchy in which one derived class inherits from multiple base class or more.

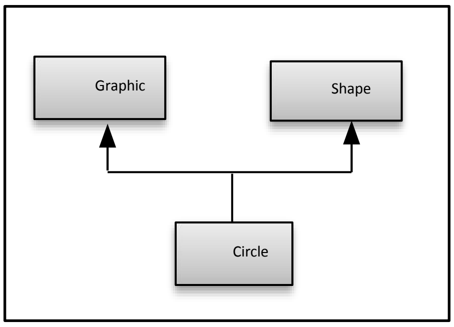


Figure (12) Multiple Inheritance

B. Metrics of inheritance

The inheritance metrics give us information about the inheritance tree of the system.

10. NEW PROPOSED INHERITANCE METRIC: APPROACH

In our approach, the estimation of class coupling is based on direct inheritance in methods. It takes into account several ways of capturing the data coupling of classes.

Two steps are followed while calculating the class coupling using the new metric: In the first step, the number of inherited methods for each class in each level in the inheritance tree is calculated. In the second step, the values of inheritance based on the number of inherited methods and the number of levels of inheritance tree are calculated.

The new metric is: **Methods Inherited Coupling Metric (MICM), it's calculate as shown:**

$$\mathbf{MICM} = \frac{\mathbf{levels\_inh}}{\mathbf{NOL}}$$

Where, leveling is the number of inherited methods for each class in each level in the inheritance tree.

NOL is the number of levels of inheritance tree.

For more information about MICM metric please contact the authors.

11. THEORETICAL VALIDATION OF MICM

We are used Weyuker's properties to an evaluate the new metric, and some points considered while devoloping MICM are verified.

11.1 Weyuker's Properties:

Property 1: Noncoarseness

Given a class A another class B can always be found such that, $\mu(A) \neq \mu(B)$, where A and B are two different classes. This implies that not every class can have the same value for a metric.

Property 2:Granularity

Let c be nonnegative number. Then there are finite numbers of program with metric c such that $\mu(A) = c$.

Property 3:Non-Uniqueness

There are distinct programs P and Q as shown Figures (3.6) and (3.7), such that $\mu(P) = \mu(Q)$.

Property 4: Design Details Matter

There exist programs P and Q having same functionality but their complexities could be different, then the reusability should difficult $(\exists A)(\exists B)(A \equiv B \,\&(\mu(A) \neq \mu(B))$

Property 5: Monotonicity

When two programs are concatenated, their metric value should be greater than the metric of the individuals.

Property 6: Nonequivalence of interaction

If there are two program bodies of equal metric value which, when separately concatenated to a same third program, yield program of different metric value. For programs A, B & C. $(\exists A)(\exists B)(\exists C)(\mu(A) = \mu(B)\&\, \mu(A + C) \neq \mu(B + C))$

Property 7:Interaction among statement

This property is not consider for OOP metrics, because, this is a property which holds for the statement count, cyclmatic number, and data flow measures.

Property 8:No change on renaming

If A is a renaming of B, then $\mu(A) = \mu(B)$.

Property 9: Interaction increases complexity

This property states that the sum of the metric values of a program could be less than the metric value of the program when considered as a whole.
$(\exists A)(\exists B)(\mu(A) + \mu(B) < \mu(A + B))$.

The MICM Metric satisfies all the Weyuker's Properties**.**

11.2 The Points Considered while Devoloping MICM are Verified:

Property 1: Non- negativity

Methods Inheritance Coupling of class of an object oriented system should belong to a specified interval i.e. Coupling $C \in [0,1]$. Normalization allows meaningful comparisons between the coupling of different classes, since they all belong to the same interval.

Property 2: Normalization

Determine if the result of the metric is normalized i.e. values returned by the metric is between 0 and 1; classes with zero coupling value have perfect coupling while classes with coupling value one have high coupling.

Property 3: Null value and maximum value

The coupling of class of an object oriented system is null if there is no interaction among the methods of the classes, and it is maximum if the interaction among the methods of classes.

Property 4:Transitivity

Consider the following 3 trees T1, T2 and T3 as shown Figures (3.16); (3.17); and (3.18), coupling T1< coupling T2, and coupling T2< coupling T3, then coupling T1< coupling T3.

Property 5: Relative number of coupling interaction:

If the interaction among the classes increase, the metric should indicate higher coupling.

The relationship between coupling and cohesion are against, where cohesion is highly coupling is few, and coupling is highly cohesion is few.

The MICM Metric satisfies all the above Properties**.**

12. OVERVIEW OF THE IMPLEMENTATION

The **MICM** tool measures the quality of O.O programs written in Java Programming Language. As shown figure (13).

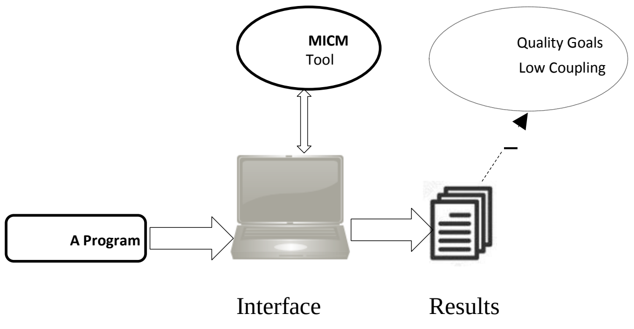


Figure (13) Overview of the Implementation

A. Flow Chart of (MICM) Tool

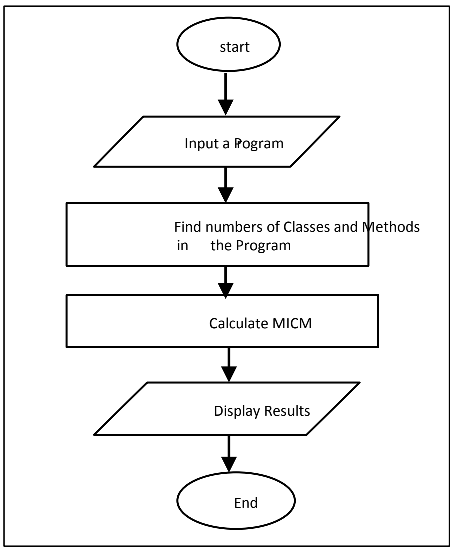


Figure (14) Flow Chart of (MICM) Tool

B. Selected Systems:

We used six OOPS in Java in this paper, that were from different websites, one project is considered default of a case study, which is rarely found in software applications, when the inheritance rate is 100%, as shown in Programs table (1).

Table (1) Details of Projects

Programs Name	No. of Classes	No. of Methods
Library management System	22	79
Airlines System	53	66
Bank Management System	46	304
Main Class	4	10
Main	29	383
Telephony Provider Master	26	398
Public Project	3	5

C. Results and Analysis

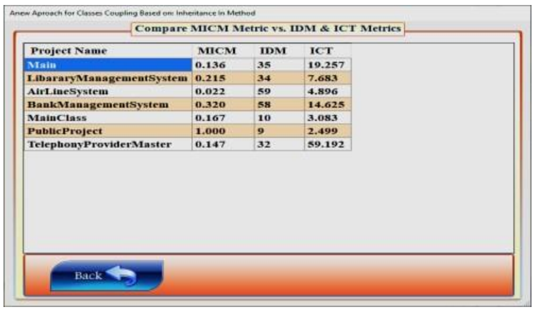
In this section we present the analysis of the results using MICM tool, and compare them it with IDM (metric), and ICT.

i. Inheritance Design Metric (IDM)[9]:

IDM metric is the values of the maximum depth of the inheritance tree ignoring any shorter paths in case of the occurring of multiple inheritance, and the Width Inheritance Tree which is number of nodes of the inheritance tree in each level.

ii. Inheritance Complexity of Tree (ICT) [10]:

ICT metric is the metric value for the entire tree and inheritance complexity of class of an inheritance tree.


Anew Aproach for Classes Coupling Based on: Inheritance in Method

Compare MICM Metric vs. IDM & ICT Metrics

Project Name	MICM	IDM	ICT
Main	0.136	35	19.257
LibararyManagementSystem	0.215	34	7.683
AirLineSystem	0.022	59	4.896
BankManagementSystem	0.320	58	14.625
MainClass	0.167	10	3.083
PublicProject	1.000	9	2.499
TelephonyProviderMaster	0.147	32	59.192

Figure (15) Comparison Between Three Metrics

As shown in Figure (15) the results of comparison between MICM, IDM, and ICT metrics. Through this explanation we result that the above figure proves that MICM metric tool always has low coupling. The result states that MICM metric always returns low values comparing to ICT and IDM metrics. Moreover, coupling refers to the degree of the relatedness of the components between two classes. Low coupling is a desirable property of software components. It’s widely recognized that lowly coupling components tend to have low maintainability and reusability.

From related works as mentioned, there are no fined minimum and maximum values to IDM and ICT metrics.

13.CONLOGIN AND FUTURE WORK

The spread of computing has led to dependence on it in the workflow of different areas of life. Therefore, when the computer program is bad it leads to poor results such as financial loss, human loss and time delay. So, software systems should work to avoid these problems. The increasing demand for software quality provides the "quality" advantage as an important factor in evaluating a software product.

MICM Metric does not take indirect inherited methods, and inherited attributes into account. For this reason, continuation of empirical works and the ability to use indirect inherited methods and inherited attributes in calculations suggested as future work.

REFERENCES

1. A. E. Abdulhafid Shuwehdi, Y.A.s., *An Investigation of Cohesion Metrics.* of World Symposium on Computer Networks and Information Security, 2014.
2. Prof. Ram Meghe , B., Maharashtra, <*Research Paper on Object-Oriented Programming (OOP).pdf*>. International Research Journal of Engineering and Technology (IRJET), 2020. **07**(10 | Oct 2020).
3. Armstrong, D.J., *The Quarks of Object-Oriented Development.* Communications of the ACM, 2006. **49**(2).
4. Weyuker, E.J., *Evaluating Software Complexity Measures*. Forgotten Books 2013, January 1985.
5. Larmen, C., *Applying UML and Patterns*. 1980.
6. Maryland, G., *Software Quality Metrics for Object Oriented System Environments.* Goddard Space Flight Center JUNE 1995.
7. Fenton, S.L.P.N.E., *Software Metrics a Rigorous and Practical Approach.* PWS Publishing Company, 08-2019.
8. Alhadi, T.A., *A New Approach for Class Cohesion Based On: Attributes Usage and Methods of Invocation.* the Libyan Academy, 2011-2012.
9. Rajnish, S.M.a.K., *New Quality Inheritance Metrics for Object-Oriented Design.* International Journal of Software Engineering and Its Applicationsol 2013. **7**: p. 6.
10. Mal, K.R.S., *Applicability of Weyuker's Property 9 to Inheritance Metric.* International Journal of Computer Applications, March 2013. . **66**: p. (0975 – 8887).