# PLHIM: Proposing an Inheritance based Object-Oriented Metric

**Latika Kharb, Deepak Chahal**

***Abstract*: Software metrics has been utilized to evaluate inheritance as well as to assist the designer in order to focus on product quality as well as cost estimation in all the lifecycle stage of development of the final product. To pertain measurement through the diverse level of class hierarchy, a person can evaluate inheritance with reuse, to acquire the best computation of abstraction levels of a object oriented system. In our paper, a new metric of hierarchical inheritance is proposed that measures the quality of the program through different levels of Object-Orientedness, and we named it PLHIM: Per Level of Hierarchical Inheritance Metric. The main idea behind proposed metrics and research work was to make use of measurement as a criterion for improvement in software development at different levels to minimize risk and this has been done by taking the problems of C++ and Java.***

***Index Terms*: Abstraction, inheritance metric, object-oriented metrics, PLHIM, reuse, software development.**

## I. INTRODUCTION

Software metrics were introduced in order to predict the programs complexity and to analyze the efforts for software development by conveying an crucial role in improving analyzing and software quality [1]. Software metric not only facilitate the designer to deal with product quality and cost estimation across all lifecycle phases of development of the final product but also enable programmers to make use of previously defined objects incorporating variables, different methods and functions through inheritance. Thus, it's not effortless to measure the acceptability or unacceptability aspect of a measurement to any particular and definite cause. An extensive variety of OO metrics have been developed to assist the software developers for growing the rates of software features like reusability, correctness and maintainability and also measuring the quality of software code. Since the object-oriented technology has entered into software organizations, it has opened new challenges for the software researchers and software organizations, who used software metrics as tools for monitoring and maintaining the software products. The main emphasis of this paper would be emphasized upon how one could minimize risk and improve software quality with the inheritance metrics. Inheritance being a useful abstraction technique, shows the relationship between design types, increase the effectiveness of a software system and reduce the complexity by lessening the quantity of operators and operations. The research paper is standardize in the following manner. Firstly, a quick view on the brief introduction of object-oriented metrics in lieu of inheritance has been explained. In the second section, a brief summary of existing inheritance based object oriented metrics has been given. The two significant inheritance metrics of Chidamber and Kemerer, also called as C&K metrics and Lorenz & Kidd set base of our research paper. The third section provides an overview of the newly proposed metric (PLHIM) based on hierarchical inheritance followed by the equation of the proposed metric. Lastly, explanation and results of PLHIM by applying proposed metric with programs in C++ and JAVA and the values with diagram and result in the table form of new proposed metric is evaluated with the benefits of proposed inheritance metric (PLHIM).

## II. OBJECT-ORIENTED INHERITANCE & ITS METRICS

Object-oriented metrics has proven to be a important tool in helping software engineers to build up large and difficult software systems as it evaluates the complexity, usability, testability and efficiency of the applications to produce better and cheaper software. By applying measurement through the different levels of hierarchy, one could evaluate inherited relations as well as reuse of the ontology [2]. Software metrics not only guide the designers to take account of cost evaluation and product excellence through all stages of final product development but also enables programmers to make use of formerly declared objects which includes variables, methods and functions through inheritance. Inheritance drops the complexity by minimizing the measure of operations as well as the operators used and side by side abstraction makes it rigid to design and maintain. Numeral inheritance based object oriented metrics are widely used over the last decade. Some significant and accessible viewpoint by different researchers on the inheritance oriented object metrics will be discussed here.

**Revised Manuscript Received on July 10, 2019.**

**Latika Kharb**, Professor, Jagan Institute of Management Studies, Delhi, India

**Deepak Chahal**, Professor, Jagan Institute of Management Studies, Delhi, India

**Table 1: Some Inheritance Metrics**

| Metrics | Formula | Description |
|---|---|---|
| **Chidamber & Kemerer's (C&K) Inheritance Metrics** | | |
| Number of Children(NOC) | NOC=Number of direct sub classes of a class | The greater the no. of children, greater the parent abstraction. |
| Depth of Inheritance Tree (DIT) | DIT= Maximum length from a node to root/base class | DIT is the greatest length of the path from the class to the starting place of the hierarchical tree. |
| **Lorenz & Kidd's Inheritance Metrics** | | |
| Number of Method Added (NMA) | NMA= Number of method added by a subclass | Larger the count of NMA, the functionality of that class becomes increasingly distinct from that of the parent classes. |
| Number of Methods Override (NMO) | NMO= Number of inherited methods overridden by a subclass | A large number of overridden methods indicate a design problem. |
| Specialization Index (SIX) | SIX= Product of the number of overridden methods and the class hierarchy nesting level normalized by the total number of method in class. | Higher the value for SIX, lesser the abstraction among its classes. |
| **Commonly Used Metrics** | | |
| Number of Multiple Inheritance (NMI) | | NMI counts the amount of instance of multiple inheritance in the system. |
| Number of Hierarchies (NOH) | | NOH counts the amount of class hierarchies in the system. |
| Number of Single Inheritance (NSI) | | NSI is sum of the number of classes that utilize inheritance in the system. |
| Average Depth of Inheritance (ADI) | | ADI is average depth of inheritance of class in the system. |
| Average Width of Inheritance (AWI) | | AWI is the average number of children/class in the system. |
| Average Number of Ancestors (ANA) | | ANA determines the average no. of classes from which a class inherit information. |
| Number of Ancestors (NOA) | | NOA is the count of the amount of distinctive classes that a class inherits. |

## II. New Proposed Metric : Per Level of Hierarchical Inheritance Metric (PLHIM)

In this research paper, we have proposed a new inheritance metric in Equation *(i)* for measuring inheritance at different phases; it's calculated by taking the mean of breadth of inheritance tree (as we calculate NOC) and depth of inheritance tree(as we calculate DIT) at each level.

**Equation of PLHIM:** *(i)*

PLHIM = Mean [(Breadth of Inheritance Tree) + (Depth of Inheritance Tree)] Per Level

*(Breadth of Inheritance is calculated as we measure Depth of inheritance and NOC metric is calculated as we measure DIT metric)*

## III. Evaluation of the PLHIM

The basic concept behind the proposed metric and how it has to be applied is demonstrated here with the help of an example in C++ with flow diagram in figure1. In this figure, a flow diagram is used to describe the problem in C++, in which the problem by hierarchical inheritance is taken. PUBLICATION class is the main or superclass. In the PUBLICATION superclass, two methods getdata() and putdata() is taken for input and output of information including title, price, page, publication, year, and editor for class PUBLICATION. From this superclass of PUBLICATION, three subclasses are developed namely, BOOK, MAGAZINE and JOURNAL. These subclasses became classes since they had children object that were subclasses. BOOK subclass contains methods, getauthor() and putauthor() for input and output of information about author and this BOOK subclass is the superclass for two subclasses TEXTBOOK and REFBOOK.

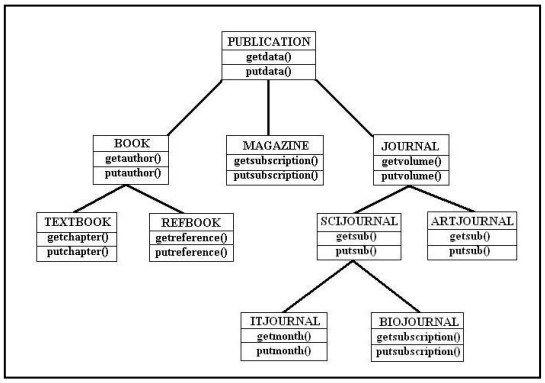


**Figure 1: Flow diagram of Hierarchical Inheritance**

In the TEXTBOOK subclass, the two methods of getchapter() and putchapter() is used to get information about the number of chapters in textbook. In the REFBOOK subclass, two methods of geteference() and putreference() are taken to get information about the number of references in the reference book. Another subclass is MAGAZINE with methods getsubscription() and putsubscription(). MAGAZINE class is the children class for superclass PUBLICATION as it does not have any subclasses. JOURNAL subclass contains two methods of putvolume() and getvolume() and contains information about the volume of journal. The JOURNAL subclass is superclass for two subclasses SCIJOURNAL and ARTJOURNAL. The SCIJOURNAL has subclasses ITJOURNAL and BIOJOURNAL. The ITJOURNAL subclass has methods of getmonth() and putmonth() for the month of publication details. The BIOJOURNAL subclass has methods of getsubscription() and putsubscription() for the subscription details. All above mentioned details could be understood easily through figure 1.

## IV. Analyzing PLHIM

The proposed metric of PLHIM could be calculated by taking the mean of breadth of inheritance tree and depth of inheritance tree per level of inheritance tree (at each level of inheritance). For evaluating our metric, we have taken problems of C++ and JAVA, but in this section we are proposing our metric with an example of C++ (figure. 1) which is described here through a flow diagram in figure 1 & 2.

Based on the concept of hierarchical inheritance, firstly depth of inheritance tree and breadth of inheritance tree at each level of tree hierarchy has to be calculated.

Then, the PLHIM could be calculated by taking the values of breadth of inheritance tree and depth of inheritance tree levelwise. Breadth of inheritance tree, as stated before is the maximum length of the path from the class to the root of the hierarchical tree i.e. same as NOC.
The depth of inheritance tree was found levelwise,In case of PLHIM (in the problems of C++ and JAVA). At level 1, 2 and 3, it was found to be 1, 2, and 3 respectively. Secondly, breadth of tree of inheritance at each level of tree hierarchy has been calculated.

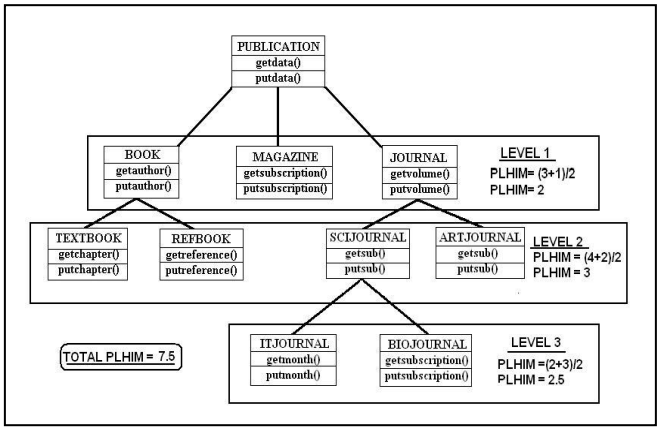


**Figure 2:Hierarchical flow diagram for PLHIM evaluation**

In the problem related to C++, breadth of inheritance tree is also found levelwise. At level 1, 2 and 3, it was found to be 3, 4 and 2 respectively in figure 2. Placing the values of breadth of inheritance tree and depth for inheritance tree at each level in proposed equation, one could get the mean of breadth of inheritance tree and depth for inheritance tree by using the PLHIM equation. This provides the value of PLHIM, as shown in figure. 2. By using this metric, classes are identified early, it could be easily redefined & redesigned i.e., it supports the concept of reusability, thus helping in early detection of erroneous data through the different level of classes. In the above problem of C++, three levels of hierarchy are taken. At level 1 (figure 2), there were three classes: BOOK, MAGAZINE and JOURNAL. By computing the PLHIM at level 1, the values of breadth for inheritance tree and depth for inheritance tree (calculated earlier) were computed in the PLHIM metric equation.

PLHIM = Mean (Breadth of Inheritance Tree (NOC) + Depth of Inheritance Tree)) Per Level

PLHIM: (3+1)/2 =2

So, at level 1, value of PLHIM is 2.

At level 2, there were four classes namely, TEXTBOOK, REFBOOK, SCIJOURNAL and ARTJOURNAL.

PLHIM: (4+2)/2 = 3

Now, after computing the PLHIM at level 2, the value of PHLIM comes out as 3.

At level 3, two classes were there, ITJOURNAL and BIOJOURNAL and after computing the PLHIM

PLHIM : (2+3)/2 = 2.5
Now at level 3, value of PHLIM was found as 2.5.

The mean of breadth and depth of inheritance tree at all the levels of inheritance separately is taken of each problem and then sum of all is made to find the value of overall PLHIM of problem which is

Total PLHIM: 2+ 3 + 2.5 = 7.5

By this total PLHIM we can easily find the complexity of program. The complexity of inheritance tree increased with increasing depth and breadth. So, instead of measuring the two values separately, if they were calculated through PLHIM metrics, it would help not only in estimating an approximate value for the tree height at each level, but also depict the whole complexity at each hierarchical level of tree separately.

We have given below result of our proposed metrics in table form after applying our metric on problems of C++ and JAVA. We have taken here three problems of C++ and three problems of JAVA. But in this paper, we have described in detail only one problem of C++ with example and diagram in this paper. We have taken in table 1, first three problems P1, P2 and P3 of C++ and P4, P5 and P6 of JAVA. This proposed metric helps in measuring complex software systems by measuring breadth and depth of inheritance tree metrics at different levels of inheritance to measure the quality and reusability. This metric helps in reducing the number of operations at different levels of inheritance.

| Metrics / Projects | | DIT | NOC | PLHIM |
|---|---|---|---|---|
| P_1 C++ | Level_1 | 2 | 3 | (2+3)/2=2.5 |
| | Level_2 | 3 | 4 | (3+4)/2=3.5 |
| | Level_3 | 4 | 2 | (4+2)/2 =3.0 |
| TOTAL PLHIM OF P_1 | | | | 9.0 |
| P_2 C++ | Level_1 | 2 | 3 | (2+3)/2=2.5 |
| | Level_2 | 3 | 5 | (3+5)/2=4.0 |
| | Level_3 | 4 | 2 | (4+2)/2 =3.0 |
| TOTAL PLHIM OF P_2 | | | | 9.5 |
| P_3 C++ | Level_1 | 2 | 5 | (2+5)/2=3.5 |
| | Level_2 | 3 | 6 | (3+6)/2=4.5 |
| | Level_3 | 4 | 4 | (4+4)/2 =4.0 |
| TOTAL PLHIM OF P_3 | | | | 12.0 |
| P_4 JAVA | Level_1 | 2 | 4 | (2+4)/2 =3.0 |
| | Level_2 | 3 | 5 | (3+5)/2 =4.0 |
| | Level_3 | 4 | 5 | (4+5)/2 =4.5 |
| TOTAL PLHIM OF P_4 | | | | 11.5 |
| P_5 JAVA | Level_1 | 2 | 3 | (2+3)/2=2.5 |
| | Level_2 | 3 | 6 | (3+6)/2 =4.5 |
| | Level_3 | 4 | 7 | (4+7)/2 =5.5 |
| TOTAL PLHIM OF P_5 | | | | 12.5 |
| P_6 JAVA | Level_1 | 2 | 3 | (2+3)/2=2.5 |
| | Level_2 | 3 | 6 | (3+6)/2=4.5 |
| | Level_3 | 4 | 9 | (4+9)/2=6.5 |
| TOTAL PLHIM OF P_6 | | | | 13.5 |

*Table-1: Results of PLHIM on the problems of C++ and JAVA*

Inheritance is a form of reusability, so this metric will also help in reusability of software. So, by calculating the PLHIM at different levels and then taking the sum of all the PLHIM at different levels of a program; we can easily find out the complexity, quality and reusability of software. In P_1, P_2 and P_3 problems of C++, we first looked for the levels of hierarchy and then at each level, we calculated the PLHIM through the values taken depthwise and breadthwise i.e. calculated the number of methods and children inherited. After finding the metrics at each level of whole problem, make a total of all the PLHIM taken at each level. Similarly, calculation for measurement of PLHIM is taken in P_4, P_5 and P_6 of JAVA problems.

## V. BENEFITS OF PROPOSED PLHIM IN SOFTWARE DEVELOPMENT AND MEASUREMENT

Metrics assess the internal/external structure, relationships, and functionality of software components [3]. The object-oriented metric evaluates the complexity, usability, testability and efficiency of the applications to produce better and cheaper software. There are abundant advantages that could be achieved by using our PLHIM metric. The developer could easily create a hierarchical structure and then find the differences directly after testing their algorithms. The Metrics value is dominated by various interrelated factors, like problem domain and solution, the objectives and goal defined for a product apart from tools and techniques in use in addition to the people who create the product. The main idea behind the research was to make use of measurement as a criterion for improvement in software development. A class located deeper in a class lattice is more fault-prone because the class inherits a large number of definitions from its ancestors [4]. Moreover, deep hierarchies imply problems of conceptual integrity, i.e. it becomes unclear which class to specialize from in order to include a subclass in the inheritance hierarchy [5]. The proposed metric for hierarchical inheritance i.e. PLHIM has been designed to aid in understanding the complexity of a class in a system and above two problems of [6] ,[4]. It help to review the details of a class at module level and also provide information about the importance of classes & methods, objects with complexity of classes and methods in terms of hierarchical inheritance. Inheritance as well as reuse could be assessed by implementing the measurement throughout for all levels of class hierarchy and larger quantity of abstraction level inherent in OO systems can be sorted. PLHIM provides easy measurement of class dependency. Moreover, when identified early, complex classes could be easily redefined & redesigned i.e. it supports the concept of reusability. It helps in early detection of erroneous data through the metric evaluation at different levels of a class. Implementation of this metric, may require more testing for those area of application and we could identify the area that have to be redesigned. Using PLHIM, various potential flaws in design could be identified and dealt with previous in software development cycle of an application.

## VI. FUTURE WORK AND CONCLUSION

By having past experiences, potential development can be guided by, to confirm the products development that always meet up quality standards and goals. In this paper, several object-oriented inheritance metrics are discussed. Our main aim is to emphasize that the proposed metric (PLHIM) differs in conditions of pleasing design features from accessible approaches. PLHIM provides easy measurement of class dependency and reusability. Moreover, when identified early, complex classes could be simply redefined and redesigned with our proposed metrics at each level of inheritance.

## REFERENCES

1. Kharb et al, “Evolution of Software Metrics: From Traditional to Object-Oriented Paradigm”, IJIT, Vol. II, Issue 4, December 2006.
2. D. Sathya and K. R. Uthayan, "Proposal for semantic metric to assess the quality of ontologies," 2011 International Conference on Signal Processing, Communication, Computing and Networking Technologies, Thuckafay, 2011, pp. 754-756. doi: 10.1109/ICSCCN.2011.6024651
3. Bansiya J, et al, “Automated Metrics and Object-Oriented Development: Using QMOOD++ for Object-Oriented Metrics”; Dr. Dobb’s Journal, 1997.
4. Basili, V. et al, “ A Validation of Object-Oriented Design Metrics as Quality Indicators”, IEEE Transactions on Software Engineering, Vol.22, 1996.
5. Daly, J. Brooks et al,” The Effect of Inheritance Depth on the Maintainability of Object-Oriented Software”, Empirical Software Engineering: An International Journal, Vol. 1, 1996.
6. Pressman R., “Software Engineering: A Practitioner’s Approach., McGraw Hill, 1997

## AUTHORS PROFILE

Dr. Latika Kharb is working in Department of MCA as GGSIPU Faculty in JIMS: Jagan Institute of Management Studies, New Delhi. She has published over 85 research papers/ articles/ chapters in Peer-reviewed / Indexed International Publishers.

Dr. Deepak Chahal is working in Department of MCA as GGSIPU Faculty in JIMS: Jagan Institute of Management Studies, New Delhi. He has published over 25 research papers. He has been the convener of many Springer CCIS Conferences.