New Quality Inheritance Metrics for Object-Oriented Design

Sandip Mal¹ and Kumar Rajnish²

¹Research Scholar, Department of Computer Science & Engineering
²Assistant Professor, Department of Information Technology
¹Sandip.mal1987@gmail.com, ²krajnish@bitmesra.ac.in

Abstract

Metrics are used to help a software engineer in quantitative analysis to assess the quality of the design before a system is built. Object-Oriented (OO) design is becoming more popular in software development environment and OO design metrics are essential parts of software environment. The primary purpose of this paper is to analytically evaluate against the Weyuker’s property and empirically validate, a proposed inheritance metrics (against a three versions of the same project) that can be used to measure the quality (especially focus on the quality factors “Reuse” and “Design Complexity”) of an OO systems in terms of the using class inheritance tree.

Keywords: Classes, Inheritance tree, Metrics, Object-Oriented, Quality Metrics

1. Introduction

Inheritance is one of the main features of OO programming paradigm. By organizing classes into a “classification hierarchy”, it gives an extra dimension to the encapsulation of the abstract data types because it enables classes to inherit attributes and methods from other classes. The inherited class can then add extra attributes and/or methods of it own [1]. Inheritance means that one class inherits that characteristic of another class as part of its definition.

Almost all researchers note the need to measure inheritance structure in terms of depths and class density. This can be measured as the depth of each class within its hierarchy, since this is likely to affect the distribution of inherited features. Briand et al. [2] empirically explored that the depth of a class in its inheritance hierarchy appears to be an important quality factor.

Chidamber and Kemerer (CK) [3] proposed the DIT metric, which is the length of the longest path from a class to the root in the inheritance hierarchy and the NOC metric, which is the number of classes that directly inherit from a given class. Henderson-Sellers [4] suggested the AID (average inheritance depth) metric, which is the mean depth of inheritance tree and is an extension of CK’s DIT. Li [5] suggested the NAC (number of ancestor classes) metric to measure how many classes may potentially affect the design of the class because of inheritance and NDC (number of descendent classes) metric to measure how many descendent classes the class may affect because of inheritance. Li [5] also theoretically validated CK metrics using a metric evaluation framework proposed by Kitchenham et al., [6] and discovered some of the deficiencies of CK[2] metrics in the evaluation process and proposed a new suite of OO metrics that overcome these deficiencies. Tegarden et al. [7] proposed the CLD (class-to-leaf depth) metric, which is the maximum number of levels in the hierarchy that are below the class and the NOA (number of ancestor) metric, which is the number of classes that a given class directly or indirectly inherits from. Lake and Cook [8]
suggested the NOP (number of parents) metric, which is the number of classes that a given
class directly inherits from and the NOD (number of descendants) metric, which is the
number of classes that directly or indirectly inherit from a class. Alshayeb et al., [11]
empirically validated two different software processes. Agarwal et al., [12-13] described the
approach of empirical study of OO metrics and presented OO design metrics.

Research has also been conducted regarding class inheritance metrics by Rajnish et al., in
[14-21]. However, it is agreed that the deeper the inheritance hierarchy, the better the
reusability of classes, making it harder to maintain the system. The designers may tend to
keep the inheritance hierarchies shallow, discarding reusability through inheritance for
simplicity of understanding [2].

The aim of the work we are presenting here is to evaluate the proposed inheritance metrics
analytically against the Weyuker’s property for analyzing the nature of the metric and
empirically to find out how the proposed inheritance metric effects on the quality factors
“reuse” and “design complexity” of an OO systems in terms of the using class inheritance
tree.

The rest of the paper is organized as follows: Section 2 presents the Weyuker’s property.
Section 3 deals with Results, which contains the description of the proposed metrics,
analytical evaluation against Weyuker properties, and discussion. Section 4 presents the
conclusion and future scope respectively.

2. Weyuker’s Property

The basic nine properties proposed by Weyuker [9] are listed below. The notations used
are as follows: P, Q, and R denote combination of classes P and Q, µ denotes the chosen
metrics, µ (P) denotes the value of the metric for class P, and P≡Q (P is equivalent to Q)
means that two class designs, P and Q, provide the same functionality. The definition of
combination of two classes is taken here to be same as suggested by [1], i.e., the combination
of two classes results in another class whose properties are union of the properties of the
component classes. Also, combination stands for Weyuker’s notion of “concatenation”.

Property 1. Non-coarseness: Given a class P and a metric µ, another class Q can always be
found such that, µ(P)≠ µ(Q).

Property 2. Granularity: There are a finite number of cases having same metric value. This
property will be met by any metric measured at class level.

Property 3. Non-uniqueness (notion of equivalence): There can exist distinct classes P and
Q such that, µ(P)= µ(Q).

Property 4. Design details are important: For two class designs, P and Q, which provide
the same functionality it does not imply that the metric values for P and Q will be same.

Property 5. Monotonicity: For classes P and Q the following must hold: µ (P) ≤ µ (P+Q)
and µ (Q) ≤ (P+Q) where P+Q imply combination of P and Q.

Property 6. Non-equivalence of interaction: ∃ P, ∃ Q, ∃ R such that, µ (P) = µ (Q) does
not imply µ (P+R) = µ (Q+R).

Property 7. Permutation of elements within the item being measured can change the metric
value.

Property 8. When the name of the measured entity changes, the metric should remain
unchanged.

Property 9. Interaction increases complexity: ∃ P and ∃ Q such that: µ (P) + µ (Q) < µ
(P+Q).

Weyuker’s list the properties has been criticized by some researchers; however, it is widely
known formal approach and serves as an important measure to evaluate metrics. In the above list however, property 2 and 8 will trivially satisfied by any metric that is defined for a class. Weyuker’s second property “granularity” only requires that there be a finite number of cases having the same metric value. This metric will be met by any metric measured at the class level. Property 8 will also be satisfied by all metrics measured at the class level since they will not be affected by the names of class or the methods and instance variables. Property 7 requires that permutation of program statements can change the metric value. This metric is meaningful in traditional program design where the ordering of if-then-else blocks could affect the program logic and hence the metric. In OOD (Object-Oriented Design) a class is an abstraction of a real world problem and the ordering of the statements within the class will have no effect in eventual execution. Hence, it has been suggested that property 7 is not appropriate for OOD metrics.

Analytical evaluation is required so as to mathematically validate the correctness of a measure as an acceptable metric. For example, Properties 1, 2, and 3 namely Non-Coarseness, Granularity, and Non-Uniqueness are general properties to be satisfied by any metric. By evaluating the metric against any property one can analyze the nature of the metric. For example, property 9 of Weyuker will not normally be satisfied by any metric for which high values are an indicator of bad design measured at the class level. In case it does, this would imply that it is a case of bad composition, and the classes, if combined, need to be restructured. Having analytically evaluated a metric, one can proceed to validate it against data.

Assumptions. Some basic assumptions used in Section 3.4 under Section 3 have been taken from Chidamber and Kemerer [3] regarding the distribution of methods and instance variables in the discussions for the metric properties.

Assumption 1: In general, two classes can have a finite number of “identical” methods in the sense that a combination of the two classes into one class would result in one class’s version of the identical methods becoming redundant. For example, a class “foo_one” has a method “draw” that is responsible for drawing an icon on a screen; another class “foo_two” also has a “draw” method. Now a designer decides to have a single class “foo” and combine the two classes. Instead of having two different “draw” methods the designer can decide to just have one “draw” method.

Assumption 2: the inheritance tree is “full”, i.e., there is a root, intermediate nodes and leaves. This assumption merely states that an application does not consist only of standalone classes; there is some use of sub classing.

3. Results

This section presents the description of the proposed quality metrics of an OO design for inheritance tree, analytical evaluation of proposed metric against Weyuker properties and discussion. First quality metric named “Inheritance Design Metric (IDM)” is measuring the width and depth of an inheritance tree and second one is measuring the total inherited classes directly/indirectly in an inheritance tree i.e. finding Inheritance Factor (IF) of the inheritance tree.

3.1. Inheritance Design Metric (IDM)

IDM is defined as follows:
IDM = WIT + D -1
Where,
WIT is the Width Inheritance Tree which is used to measure the width of the tree and can be defined as:
\[ WIT = \text{Max} \{\text{number of nodes of the inheritance tree in each level}\} \]
D represents the maximum depth of the inheritance tree ignoring any shorter paths in case of the occurring of multiple inheritance.

**Viewpoints.**
- More width means large number of children which are considered to be difficult to modify and usually require more testing because of the effects on the changes which happens to all children classes that has a larger number of children which may require more testing of the methods of that class and thus testing time will increase.
- More WIT promotes less reuse and increase design complexity of classes and less WIT promotes more reuse and less complex (but somehow effects on the design complexity of classes due to inheritance) during development phase.
- Since D consider better than WIT in the inheritance tree because it promotes reuse of methods and attributes through inheritance.

At the design stage some mental exercise is required for designer to frame out whether to increase or decrease the D and WIT according to the system requirements or if any modification is required to control the overall architecture of the system.

### 3.2. Normalized Inheritance Factor (NIF)

Another important part which we have considered about inheritance tree is “Inheritance Factor (IF)” that indicates how many classes in the inheritance tree is directly / indirectly inherited. It is calculated as follows:

\[
IF(C_i) = \left\{ \sum_{i=1}^{L} \left( \frac{\text{TIDC}(C_i)}{\text{TIIC}(C_i) + 1} \right) \right\}
\]

Where, TIDC (C_i) represents the Total Inherits Direct Classes at the i\textsuperscript{th} level in the inheritance tree.

TIIC (C_i) + 1 represent the Total Inherits Indirect Classes at the i\textsuperscript{th} level in the inheritance tree. Further new classes may be added at the last level of the inheritance tree so for the completeness we had taken +1 with TIIC (C_i).

Since IF is not normalized metric, so to relate IDM (in view of WIT and D) with IF we define Normalized Inheritance Factor (NIF) as shown below:

NIF (for entire tree) = \( \frac{\sum IF}{t} \) \( \in [0, 1] \).

Where, t represents the total number of classes in the inheritance tree.

### 3.3. Relationship between NIF and IDM

This section presents the relationship between the Quality Metrics (NIF and IDM). The primary purpose of this quality metrics is “to give an indication of amount of reuse (generally reuse also relates to understandability and Testability) of classes in the inheritance tree” and “find out how the design complexity is high/less”. Both quality metrics are used to measure the inheritance tree. After verifying the results from same project of different versions (Version1.0, Version 2.0 and Version 3.0), certain hypothesis is provided for measuring the quality factors (especially focus on reuse and design complexity) of an OO design systems

\[ H_1: \text{if } NIF < 0.5 \text{ then } \]

Increase D and reduce WIT of the inheritance tree.

See in Table I the effects on the quality factors for the Hypothesis H_1.
Table 1. Effects on the Quality Factors

<table>
<thead>
<tr>
<th>Reuse</th>
<th>Understandability</th>
<th>Design Complexity</th>
<th>Testability</th>
</tr>
</thead>
<tbody>
<tr>
<td>More</td>
<td>More (In terms of clarity)</td>
<td>Increases</td>
<td>Expensive</td>
</tr>
</tbody>
</table>

$H_2$: if $NIF \geq 0.5$ then

Increase WIT and reduce $D$ in the inheritance tree.

See in Table 2 the effects on the quality factors for the Hypothesis $H_2$.

Table 2. Effects on the Quality Factors

<table>
<thead>
<tr>
<th>Reuse</th>
<th>Understandability</th>
<th>Design Complexity</th>
<th>Testability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less</td>
<td>Less</td>
<td>Increase</td>
<td>Expensive</td>
</tr>
</tbody>
</table>

From the above mentioned hypothesis and from Table 1 and Table 2 following observation made:

- Design Complexity and Testability is expensive if WIT and $D$ is high.
- Understandability is difficult and Reuse is very less if WIT is very high and vice-versa for the case of $D$.

3.4. Analytical Evaluation of IDM against Weyuker’s Properties

From assumption 2, there is a root and leave classes in the inheritance tree. Generally the depth of leaf classes always greater than the root class and the width of subclasses may or may not be the same of super classes. So, there exist a classes in the inheritance tree where $IDM(P) = IDM(Q)$ and $IDM(P) \neq IDM(Q)$. Hence, Property 1 (Non-coarseness) and Property 3. Non-uniqueness (notion of equivalence) is satisfied.

There is finite number of cases in the inheritance tree having the same IDM values for classes. Since IDM is the design metric and measured at the class level so Property 2. Granularity is satisfied.

Generally, design of classes and subclasses involves choosing what properties the class must inherit in order to perform its functions. In other words we can say that depth and width of the inheritance tree is design implementation dependent. Hence, Property 4(Design details are important) is satisfied.

When any two classes say class $P$ and class $Q$ in the inheritance tree are combined then there are three possible cases involved:

Case 1: when two classes (Class $P$ and Class $Q$) are at the same level in the inheritance tree-

![Figure 1. Before Combination of Class P and Class Q](image-url)
Figure 2. After Combination of Class P and Class Q

From Figure 1, $D(P) = D(Q) = x$ and $WIT(P) = WIT(Q) = y$ and $IDM(P) = IDM(Q) = x + y - 1$.

From Figure 2, $D(P+Q) = x$ and $WIT(P+Q) = y - 1$ and $IDM(P+Q) = x + y - 2$.

Therefore, we get $IDM(P) \geq IDM(P+Q)$ and $IDM(Q) \geq IDM(P+Q)$. Hence, Property 5 Monotonicity is not satisfied.

Case 2: when two classes (Class P and Class Q) are at the different level and class Q is the child of class P in the inheritance tree-

Figure 3. Before Combination of Class P and Class Q

Figure 4. After Combination of Class P and Class Q

From Figure 3, $IDM(P) = x$ and $IDM(Q) = y$ where $y > x$.

From Figure 4 $IDM(P+Q) = x$.

Therefore, we get $IDM(P) \leq IDM(P+Q)$ and $IDM(Q) \geq IDM(P+Q)$. Hence, Property 5 Monotonicity is not satisfied.

Case 3: when two classes (Class P and Class Q) are at the different level and class Q is not the child of class P in the inheritance tree-
If class P+Q is located as the immediate ancestor to classes U, B, R and T (P’s location in the inheritance tree, the combine class cannot inherit method from C, however if P+Q is located as an immediate child of C (Q’s location) the combined class can still inherits method from all the ancestors of P and Q, therefore, P+Q will be located Q’s location.

From Figure 5 IDM (P) = x and IDM (Q) = y where y > x. From Figure 6 IDM (P+Q) = y. Therefore, we get IDM (P) ≤ IDM (P+Q) and IDM (Q) ≤ IDM (P+Q). Hence, Property 5 Monotonicity is satisfied.

From Figure 7, Figure 8 and Figure 9, it is observed that IDM (P) = IDM (Q’) = x (see Figure 7), from Figure 8 and Figure 9, IDM (P+R) = IDM (Q’+R) = x. So, there exist three classes say P, Q’ and R such that IDM (P) = IDM (Q’) which implies IDM (P+R) = IDM (Q’+R). Hence, Property 6 Non-equivalence of interaction is satisfied.
Property 7 requires that permutation of program statements can change the metric value. This metric is meaningful in traditional program design where the ordering of if-then-else blocks could affect the program logic and hence the metric. In OOD (Object-Oriented Design), a class is an abstraction of a real-world problem and the ordering of the statements within the class will have no effect in eventual execution. Hence, it has been suggested that property 7 is not appropriate for OOD metrics.

Property 8 is satisfied because when the name of the measured entity changes, the metric should remain unchanged.

Property 9 Interaction increases complexity: \( \exists P \) and \( \exists Q \) there is always a case that IDM (P) + IDM (Q) is always > than IDM (P+Q). It is because when we combined the two classes, number of classes after combination is reduced and one more case exist i.e., WIT (P+Q) is always less than the individual values of WIT (P) and WIT (Q) (D may or may not be same). Hence, Property 9 is not satisfied. The analytical evaluation result for the new inheritance metric IDM is shown in Table 3.
Table 3. Analytical Evaluation Results for IDM against Weyuker Properties

<table>
<thead>
<tr>
<th>Property Number</th>
<th>IDM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>✓</td>
</tr>
<tr>
<td>3</td>
<td>✓</td>
</tr>
<tr>
<td>4</td>
<td>✓</td>
</tr>
<tr>
<td>5</td>
<td>×</td>
</tr>
<tr>
<td>6</td>
<td>✓</td>
</tr>
<tr>
<td>7</td>
<td>NOT APPLICABLE</td>
</tr>
<tr>
<td>8</td>
<td>✓</td>
</tr>
<tr>
<td>9</td>
<td>×</td>
</tr>
</tbody>
</table>

✓: this indicates that metric satisfy the corresponding property.
×: this indicates that metric does not satisfy the corresponding property.

3.5. Discussion

3.5.1. Data Collection and Flow of Proposed Quality Model: We had collected Java projects from experienced PG students of our university. We consider our collected java project as version 1.0 (involves 30 inheritance trees). All these projects have been modified by experienced PG students of our university. They developed version 2.0 (involves 30 inheritance trees) by adding more classes with version 1.0 and also developed version 3.0 (involves 40 inheritance trees) by adding more classes with Version 2.0. Figure 10 shows the flow of proposed quality model.

![Figure 10. Flow of Proposed Quality Model](image)

3.5.2. Empirical Data: The graphs, summary statistics and Correlation coefficients for propose inheritance metrics for all versions (version 1.0, version 2.0, and version 3.0) are shown in Figure 11, Figure 12, Figure 13 and Table 4, Table 5, Table 6, Table 7, Table 8 and Table 9.
Figure 11. Relationship of NIF with D and WIT for the Version 1.0

Figure 12. Relationship of NIF with D and WIT for the Version 2.0

Figure 13. Relationship of NIF with D and WIT for the Version 3.0
### Table 4. Summary Statistics for the Data Set Version 1.0

<table>
<thead>
<tr>
<th>OO Metrics</th>
<th>Median</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Average Percentile (25%, 50%, and 75%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>4</td>
<td>2</td>
<td>7</td>
<td>3.6666</td>
</tr>
<tr>
<td>WIT</td>
<td>4</td>
<td>2</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>IDM</td>
<td>6</td>
<td>4</td>
<td>14</td>
<td>6.6666</td>
</tr>
<tr>
<td>NIF</td>
<td>0.5125</td>
<td>0.30</td>
<td>0.88</td>
<td>0.5246</td>
</tr>
</tbody>
</table>

### Table 5. Summary Statistics for the Data Set Version 2.0

<table>
<thead>
<tr>
<th>OO Metrics</th>
<th>Median</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Average Percentile (25%, 50%, and 75%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>4.5</td>
<td>3</td>
<td>8</td>
<td>4.5833</td>
</tr>
<tr>
<td>WIT</td>
<td>6</td>
<td>2</td>
<td>17</td>
<td>5.75</td>
</tr>
<tr>
<td>IDM</td>
<td>9</td>
<td>5</td>
<td>21</td>
<td>9.3333</td>
</tr>
<tr>
<td>NIF</td>
<td>0.5164</td>
<td>0.24</td>
<td>0.67</td>
<td>0.5091</td>
</tr>
</tbody>
</table>

### Table 6. Summary Statistics for the Data Set Version 3.0

<table>
<thead>
<tr>
<th>OO Metrics</th>
<th>Median</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Average Percentile (25%, 50%, and 75%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>5</td>
<td>3</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>WIT</td>
<td>6</td>
<td>2</td>
<td>26</td>
<td>6</td>
</tr>
<tr>
<td>IDM</td>
<td>10</td>
<td>5</td>
<td>29</td>
<td>10.3333</td>
</tr>
<tr>
<td>NIF</td>
<td>0.4645</td>
<td>0.20</td>
<td>0.67</td>
<td>0.4542</td>
</tr>
</tbody>
</table>

### Table 7. Correlation Coefficients between Metrics for the Data Set Version 1.0

<table>
<thead>
<tr>
<th>Correlation Coefficient</th>
<th>IDM</th>
<th>IF</th>
<th>D</th>
<th>WIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDM</td>
<td>1.000</td>
<td>0.836</td>
<td>0.735</td>
<td>0.900</td>
</tr>
<tr>
<td>IF</td>
<td>0.836</td>
<td>1.000</td>
<td>0.422</td>
<td>0.877</td>
</tr>
<tr>
<td>D</td>
<td>0.735</td>
<td>0.422</td>
<td>1.000</td>
<td>0.366</td>
</tr>
<tr>
<td>WIT</td>
<td>0.735</td>
<td>0.877</td>
<td>0.366</td>
<td>1.000</td>
</tr>
</tbody>
</table>
Table 8. Correlation Coefficients between Metrics for the Data Set Version 2.0

<table>
<thead>
<tr>
<th>Correlation Coefficient</th>
<th>IDM</th>
<th>IF</th>
<th>D</th>
<th>WIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDM</td>
<td>1.000</td>
<td>0.886</td>
<td>0.527</td>
<td>0.947</td>
</tr>
<tr>
<td>IF</td>
<td>0.886</td>
<td>1.000</td>
<td>0.292</td>
<td>0.905</td>
</tr>
<tr>
<td>D</td>
<td>0.527</td>
<td>0.292</td>
<td>1.000</td>
<td>0.226</td>
</tr>
<tr>
<td>WIT</td>
<td>0.947</td>
<td>0.905</td>
<td>0.226</td>
<td>1.000</td>
</tr>
</tbody>
</table>

Table 9 Correlation Coefficients between Metrics for the Data Set Version 3.0

<table>
<thead>
<tr>
<th>Correlation Coefficient</th>
<th>IDM</th>
<th>IF</th>
<th>D</th>
<th>WIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDM</td>
<td>1.000</td>
<td>0.888</td>
<td>0.233</td>
<td>0.960</td>
</tr>
<tr>
<td>IF</td>
<td>0.888</td>
<td>1.000</td>
<td>0.031</td>
<td>0.918</td>
</tr>
<tr>
<td>D</td>
<td>0.233</td>
<td>0.031</td>
<td>1.000</td>
<td>0.032</td>
</tr>
<tr>
<td>WIT</td>
<td>0.960</td>
<td>0.918</td>
<td>0.032</td>
<td>1.000</td>
</tr>
</tbody>
</table>

3.5.3. Discussion: From Figure 11, Figure 12 and Figure 13, it is observed that 57% and 60% inheritance tree in Version 1.0 and Version 2.0 (see Figure 11 and Figure 12) respectively have NIF value greater than 0.5. This indicates that most of the classes in the inheritance tree are top heavy (most classes are near to the root of the tree) and designer only focus on increasing width rather than depth in these versions. In these versions designer is not taking the advantage of reuse of code, only add extra classes at some level in the inheritance tree in Version 2.0, making it more complex to predict the behavior of the projects and also difficult to understand the nature of classes. But in version 3.0 (see Figure 13) only 35% projects have NIF greater than 0.5. This indicates that most of the classes in the inheritance tree are bottom heavy (most classes are near to the bottom of the tree) by increasing depth. This indicates that designer is taking advantage of reuse of code and understandability of code (in terms of clarity) is easy (by saving codes of the projects) but somehow it will effect on the design complexity due to increase in depth of the tree.

Following interesting observations is also made from Table 4, Table 5 and Table 6 that the value of median, mean, maximum and average percentile of IDM value in all three version increases, this indicates value of D or value of WIT definitely has increased in maximum inheritance trees in higher version. Median, average percentile value of NIF is greater than 0.5 in version 1.0 and version 2.0. This shows that the designer is more focus on the width of the tree (by adding more new classes at any level in the inheritance tree) rather than depth in maximum inheritance trees of version 1.0 and Version 2.0 and there is a possibility of less reuse of code (not saving code by increasing width) and also gives an indication about the complex behavior of the projects at the design stage. But Median, average percentile value of NIF is less than 0.5 in version 3.0, this shows that the designer is more focus on the depth of the tree (by adding more new classes at any level in the inheritance tree) rather than width in version 1.0 and Version 2.0 and there is a more inheritance i.e., possibility of more reuse of code (saving code by increasing depth) and also gives an indication about the easy behavior of the projects at the design stage. Minimum value of NIF is always less than 0.5 in all three versions; this indicates there are some inheritance trees in every version that have more depth.
than width. Maximum value of NIF is always greater than 0.5 in all three versions; this indicates there are some inheritance trees in every version that have more width than depth.

Further, following interesting observations made from Table 7, Table 8 and Table 9. IDM is correlated very well with IF and WIT in all the versions (1.0, 2.0, and 3.0) of the same project (see Table 7, Table 8 and Table 9) but not correlated well with D especially in updated Version 3.0 is 0.233 (see Table 9). This shows that the designer is more focus on the width of the tree (by adding more new classes at any level in the inheritance tree) rather than depth in Version 3.0 and there is a less inheritance i.e. possibility of less reuse of code (not saving code by increasing width) and also gives an indication about the complex behavior of the project at the design stage. Similar behavior for IF with D and WIT i.e. IF is correlated very well with WIT in all the three versions; 0.875, 0.905, 0.918 (see Table 7, Table 8 and Table 9) and does not correlated well with D in all the three versions; 0.422, 0.292, 0.031 (see Table 7, Table 8 and Table 9). This shows that the changing of WIT effects the value of IDM and IF.

4. Conclusion and Future Scope

In this paper an attempt has been made to present two quality metrics one is “IDM (Inheritance Design Metric)” which is measuring the width and depth of an inheritance tree and second one is measuring the total inherited classes directly/indirectly in an inheritance tree i.e., finding Inheritance Factor (IF) of the inheritance tree. The paper is first analyzes the nature of the proposed metric through Weyuker’s property and it satisfies all the criteria as mentioned by Weyuker’s [9] (see Section 3.4). Secondly, a validation is done on the three versions of data set of the same project to assess the impact of quality factors (reuse and design complexity) upon the inheritance tree. The results of our validation suggest that (see Figure 11, Figure 12 and Figure 13) if NIF is less than 0.5 then there is increasing in depth and decreasing in WIT of the inheritance tree that indicates more reuse, more understandability of code (in terms of clarity) and design complexity is expensive. If NIF is greater than equal to 0.5 then there is increasing in WIT and reducing depth of the tree that indicates less reuse of code, difficult to understand the behavior of code and negative impact on the design complexity of the inheritance. See Table 7, Table 8 and Table 9, we found that IDM is correlated very well with WIT and IF but not correlated well with D because one of the reason may be a designer’s choice about “WIT or D”. This firmly belief us that both quality inheritance metrics are the good measure of quality for the systems using inheritance tree.

In terms of future scope we focus on some fundamental issues: (1) further validation is necessary to improve our understanding of conceptual model of quality. (2) To perform an empirical relation between proposed inheritance metrics with other existing inheritance metrics for increasing its usefulness.

References


Authors

Mr. Sandip Mal completed B.Tech from West Bengal University of Technology, India in the year 2008. He has also completed ME (Software Engineering) from Birla Institute of Technology, Mesra, Ranchi, Jharkhand, India in the year 2012. Currently, he is pursuing PhD on Software Quality Metrics. His Research area is Object-Oriented Metrics, Software Engineering, Database System, and Image Processing.
Dr. Kumar Rajnish is an Assistant Professor in the Department of Information Technology at Birla Institute of Technology, Mesra, Ranchi, Jharkahnd, India. He received his PhD in Engineering from BIT Mesra, Ranchi, Jharkhand, India in the year of 2009. He received his MCA Degree from MMM Engineering College, Gorakhpur, State of Uttar Pradesh, India. He received his B.Sc Mathematics (Honours) from Ranchi College Ranchi, India in the year 1998. He has 24 International and National Research Publications. His Research area is Object-Oriented Metrics, Object-Oriented Software Engineering, Software Quality Metrics, Programming Languages, and Database System.