JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2003**

Vol. 2, No. 3, May-June 2003

Application of a Revised DIT Metric to Redesign an OO Design

Ghassan Alkadi, Southeastern Louisiana University, USA.
Ihssan Alkadi, University of Louisiana at Lafayette, USA.

Abstract

In this paper, we continue a series of papers that discuss specific design metrics [Alkadi 1999] [Alkadi 2000] [Alkadi 2001] [Alkadi 1998]. The design metric discussed in this paper is the Depth of Inheritance [DIT] metric. Design evaluation is a recurring step that should be performed and checked multiple times before committing to the final design implementation. Metrics are utilized to evaluate inheritance and reuse in order to take into account the greater number of abstraction levels inherent in object-oriented systems. Furthermore, they facilitate the designers to address cost estimation and product quality across all life-cycle stages of developing the final product.

1 INTRODUCTION

Inheritance is a relationship among classes wherein one class shares the structure or behavior defined in one (single inheritance) or more (multiple inheritance) classes [Booch, 1991]. Single inheritance occurs when a subclass inherits behavior of some superclass. A subclass may change the behavior or structure of some superclass. Multiple inheritance occurs when a subclass inherits from multiple superclasses. Inheritance reduces redundancy in the code and thereby increases its efficiency. **DIT** is the number of ancestor classes that can affect a class. The deeper a class is in the hierarchy, the higher the degree of methods inheritance, making it more complex to predict its behavior. [Chidamber, S., Kemerer 1991] [Chidamber, S., Kemerer 1994] introduced a metrics suite for object-oriented designs. [Chidamber, S., Kemerer 1991] Chidamber, S., Kemerer 1994] formally evaluated the metrics against a widely accepted list of software metric evaluation criteria. They claimed that such measures applied in a software system could be used to aid management in:

- Estimating the cost and schedule of future projects,
- Evaluating the productivity impacts of new tools and techniques,
- Establishing productivity trends over time,

Cite this article as follows: G. and I. Alkadi: "Application of a Revised DIT Metric to Redesgin an OO Design", in *Journal of Object Technology*, vol. 2, no. 3, May-June 2003, pp. 127-134.
http://www.jot.fm/issues/issue_2003_05/article3

- Improving software quality,
- Forecasting future staffing needs, and
- Anticipating and reducing future maintenance requirements.

The inheritance hierarchy has a root and leaves. The depth of inheritance of a leaf is always greater than that of the root [Chidamber, S., Kemerer 1991]. The *DIT*(C) is the distance from class C to the root. If multiple inheritance exists, then the DIT is the longest path for the distance. It is a system-level metric that indicates how many levels of inheritance have to be investigated for evaluating the whole class hierarchy. The deeper the class, the greater the number of methods to inherit, thus making it difficult to maintain. Increased difficulty in maintenance is likely because of the introduction of more public and protected methods. In addition, the introduction of more public and private methods increases the chances of extensions and overrides, which in return increases the difficulty of testing [Lorenz, M., Kidd 1994]. [Basili, V., Briand, L., Melo 1996] introduced a hypothesis 'H-DIT' for the DIT metric. They suggested that well-designed object-oriented systems are those structured as forests of classes, rather than as one very large lattice. [Basili, V., Briand, L., Melo 1996] also suggested that a class located deeper in a class lattice is more fault-prone because the class inherits a large number of definitions from its ancestors. 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 [Daly, J., Brooks, A., Miller, J., Roper, J., Wood 1996].

2 REDESIGN USING DIT METRIC

We now define the methods to use the DIT metric as a part of the redesign process. A class hierarchy in a structure tree has a base called the root. A low number of levels in a hierarchy suggests difficulties in finding the abstractions and specializations to optimize reuse through inheritance. On the other hand, a large number of levels suggests no subclassing by specialization (is-a) [Lorenz, M., Kidd 1994].

An example of subclassing by type is shown in Figure 1. There are some car models produced by GM corporation that have similar specifications and looks but different names, for instance the Chevy Tahoe and the GMC Yukon. The Tahoe is a subclass under Chevy trucks whereas the Yukon is a subclass under GMC trucks. GMC trucks are at the same level of the Chevy object and one level above the Yukon. If we merge the Yukon truck with the Chevy Tahoe, we reduce testing and reuse code more efficiently. The new tree is depicted in Figure 2. [9] introduced a Hypothesis-DIT 'H-DIT' for the DIT metric. They suggested that well-designed object-oriented systems are those structured as forests of classes, rather than as one very large lattice.

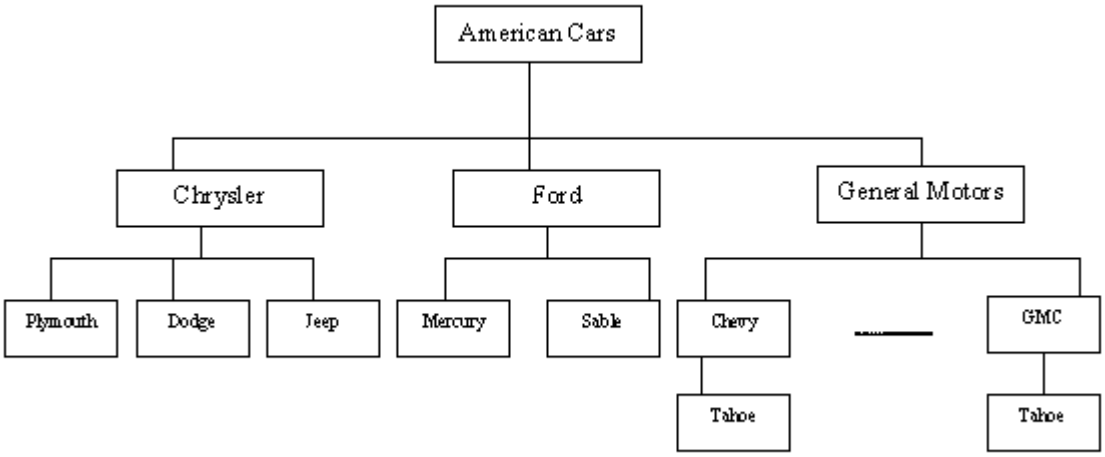


Fig. 1 Subclassing by type

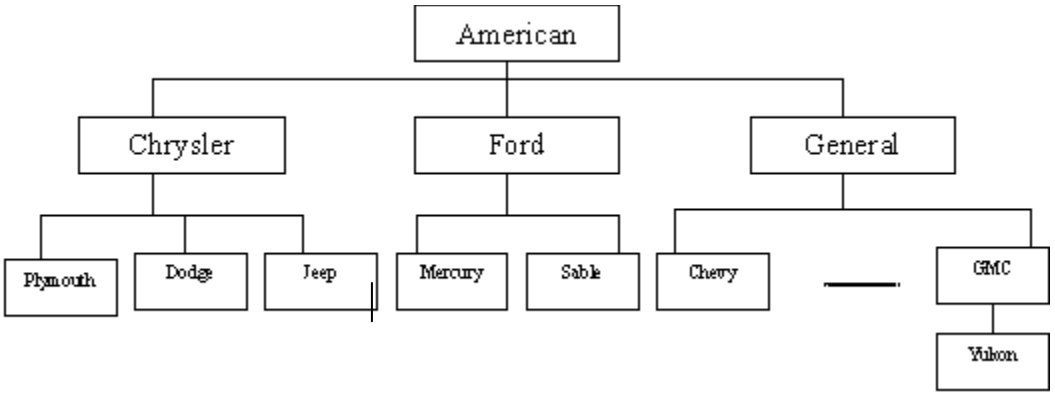


Fig. 2 Subclassing by type after using code reuse

We define a Depth of Inheritance Tree (DIT) algorithm to determine whether we need to extend the number of levels in a class hierarchy. We then introduce examples to show how the algorithm works. The algorithm is given in Figure 3. In the DIT algorithm, we use thresholds where the minimum level of a tree is **2** and the maximum is **6** levels. These thresholds are based on the recommendations of [Henderson-Sellers 1996], [Lorenz, M., Kidd 1994] and [Basili, V., Briand, L., Melo 1996].

The DIT algorithm determines if we need to extend the number of levels in a hierarchy by checking the degree of similar methods "inherited" and instance variables used among the objects in one level. After we find the objects that are most similar in the use of inherited methods; we then rank them by using the ranking factor, P_i. After we sort the percentages, we then make the second highest ranked object obtained from the sorting procedure the child of the highest ranked object. This process results in adding one level

to the hierarchy. We show an example of a hierarchy tree that is depicted in Figure 4 where the number of levels is greater than 6. Let us assume that the number of levels is 7. So, the two levels to be affected by this algorithm are only the sixth and seventh levels. Classes C_2 and C_3 are merged with class C_1 without affecting any of the levels preceding level 6 in the class hierarchy.

DIT Algorithm

Let T_{lmin} be the minimum number of levels in a class hierarchy, where $T_{lmin} \geq 2$.
Let T_{lmax} be the maximum number of levels in a class hierarchy, where $T_{lmax} \leq 6$.
Let Po_j be the percentage of inherited methods in object o_j.

$$Max\begin{bmatrix}\forall \text{ objects in a class hierarchy} \\ \text{measure path from the root}\end{bmatrix} = \text{depth of the tree } = n$$

```
Algorithm_DIT(n);
Begin
If (n < T_lmin) and number of subclasses < 2 Then
   Delete_object()          /* Check if it should exist at all in the hierarchy*/
Else
If (n < T_lmin) Then
   Rank objects at level n by similarity, using percentages of inherited methods
  PIM P_i as a ranking factor;
   Rank_objects(n);
   Make the second highest ranked object the child of the highest ranked object
   (Thus adding one level to the hierarchy tree);
Else
If (n > T_lmax) Then
   Merge all objects at level n with parent at level n-1;
   Call Algorithm_DIT(n-1);
Else
   No action required;
Endif;
Return;

Rank_Objects(n);
Begin
        A = array [1..N_n];
        for each object o_j, j=1.. N_n , where N_n = total number of objects at level n
                A[j] = Po_j;        /* Percentage of inherited methods in object o_j */
        Sort(A);            /* A_[1] = highest ranking */
        Return A;
End Rank_Objects;
End Algorithm_DIT;
```

Fig. 3 DIT Algorithm

We show another example where the number of levels is less than 2, and more than one object exists in the child's level. Figure 5 shows the class hierarchy for the example.

After we apply the DIT algorithm, we make object class C_4 the child of C_2 based on the percentage of inherited methods in each class. The class hierarchy changes in depth, which utilizes the inheritance property and recommends a depth of at least two levels.

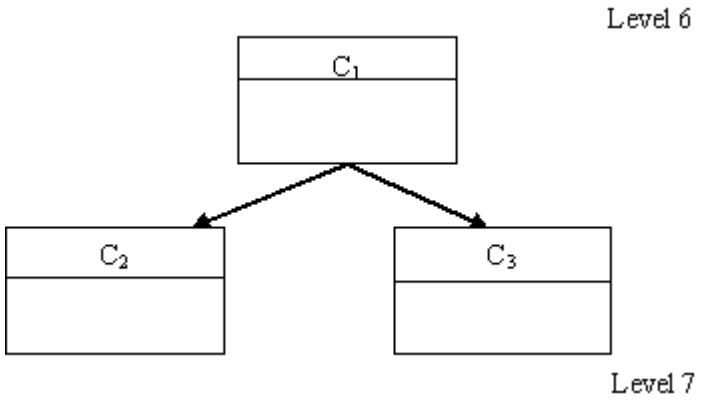


Fig. 4 Hierarchy of Depth 1

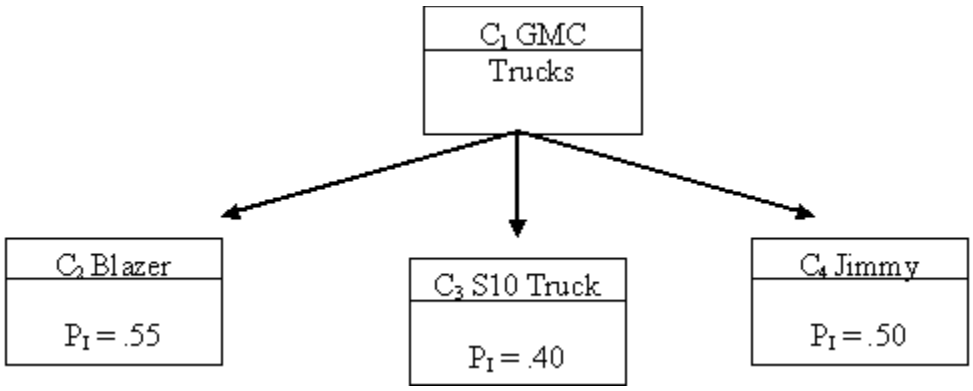


Fig. 5 Hierarchy of Depth of 1 and with more than 1 child

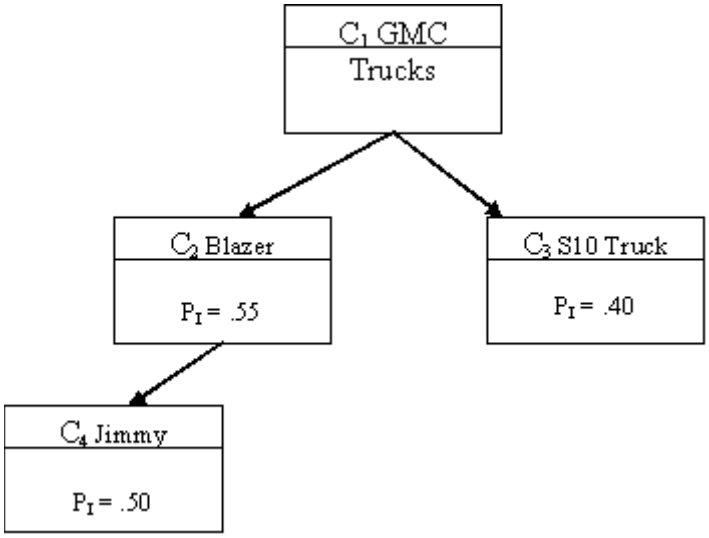
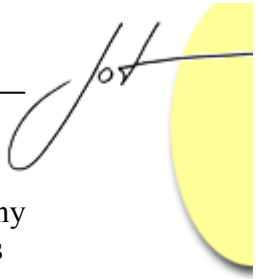


Fig. 6 Hierarchy of Depth 2 after applying DIT algorithm

Also, it should be noted that the change made is minimal and did not affect the original design. The new class hierarchy is shown in Figure 6. The hierarchy in Figure 6 shows that the inheritance characteristic is more efficiently utilized than the hierarchy in Figure 7. Moreover, rather than deleting the hierarchy in Figure 7 since it had an inheritance depth of 1, we extended the hierarchy by 1 level and therefore, justified its existence. Since half of the methods existing in class object C_4 are inherited methods, this situation will not affect the structure or the behavior of this because C_4 is still inheriting those methods from the root. The overridden and the pure methods in C_4 will remain the same without the need for changing their structure. The DIT algorithm also works in the same manner for a hierarchy that has depth larger than 6 levels.

3 SUMMARY

There are numerous advantages from using the metrics algorithms. The designer can see the difference in the hierarchy structure instantly right after he/she applies the DIT algorithm. The deeper the class, the greater the number of methods to inherit, making the class difficult to maintain. Increased difficulty in maintenance is likely because of the introduction of more public and protected methods. In addition, the introduction of more public and private methods increases the chances of extensions and overrides, which in return increases the difficulty of testing. There is greater potential reuse of inherited methods if the depth is > 2 since reuse further specializes the superclass type of object. However, we indicated that any depth > 6 is enough since more levels indicate the possibility of not subclassing by specialization (is-a) but rather implementation subclassing. Keeping this in mind, we defined the DIT algorithm that determines whether we need to extend the number of levels in a class hierarchy or reduce it. The examples



that we showed illustrated the importance of using the DIT metric. The class hierarchy changed in depth, which utilized the inheritance property with minimum design changes

REFERENCES

[Alkadi1999] Alkadi, G. "Restructuring Object-Oriented Designs Using A Metric-Driven Approach," A Dissertation submitted to the computer science department, May 1999.

[Alkadi2000] Alkadi, G., Alkadi, I. "Applying A Revised CBO Metric To Redesign an OO Design," *Proceedings of the Southern Conference on Computing*, The University of Southern Mississippi, October 26-28, 2000.

[Alkadi2001] Alkadi, G., Alkadi, I. "Applying A Revised RFC Metric to Redesign An OO Design," *IEEE Aerospace Conference*, Big Sky, Montana, March 2001.

[Alkadi1998] Alkadi, G., Carver, D. "Application of Metrics to Object-Oriented Designs," Proceedings of the 1998 *IEEE Aerospace Conference,* March 1998.

[Booch1991] Booch, G. *Object Oriented Design with Applications*, (Benjamin/Cummings Publishing Co., Inc. 1991.)

[Chidamber1991] Chidamber, S., Kemerer, C. "Towards a Metric Suite for Object Oriented Design," Sloan School of Management, MIT, *OOPSLA 91*, pp. 197-211.

[Chidamber1994] Chidamber, S., Kemerer, C. "A Metrics Suite For Object Oriented Design," *IEEE Transactions on Software Engineering*, *Vol. 20*, No. (6), June 1994, pp. 476 - 493.

[Lorenz1994] Lorenz, M., Kidd, J. *Object Oriented Software Metrics*, (Prentice Hall, 1994).

[Basili1996] Basili, V., Briand, L., Melo, W. "A Validation of Object-Oriented Design Metrics as Quality Indicators," *IEEE Transactions on Software Engineering*, *Vol.* 22, No. 10, October 1996, pp. 751 – 761.

[Daly1996] Daly, J., Brooks, A., Miller, J., Ropber, J., Wood, M. "The Effect of Inheritance Depth on the Maintainability of Object-Oriented Software," *Empirical Software Engineering*: *An International Journal*, *Vol.* 1, no, 2, February 1996.

[Henderson1996] Henderson-Sellers, B. *Object-Oriented Metrics, Measures of Complexity*, The Object Oriented Series, (Prentice Hall, 1996).

About the authors

Ghassan Alkadi is on the faculty at Southeastern Louisiana University (SLU). He received his B.S. Degree in Computer Science at SLU, May 1985. In December 1991 he earned his MS. In Systems Science fromLouisiana State University (LSU). He earned his Doctoral degree in Computer Science at LSU. His areas of expertise include software engineering in general, testing in particular, Internet, HTML, and operating systems. His research interests include testing in object oriented systems, systems Design, and programming languages. Email: galkadi@selu.edu

Ihssan Alkadi is on the faculty at University of Louisiana at Lafayette (ULL). He received his B.S. Degree in Computer Science at SLU, May 1985. In May 1992 he earned his MS. In Systems Science fromLouisiana State University (LSU). He earned his Doctoral degree in Computer Science at LSU. His areas of expertise include software engineering in general, testing in particular, Internet, HTML, and operating systems. His research interests include testing in object oriented systems, systems validation, and system verification. Email: ialkadi@louisiana.edu