Mathematical Assessment of Object-Oriented Design Quality

Alexander Chatzigeorgiou, *Member*, *IEEE*

Abstract—A method of link analysis employed for retrieving information from the Web is extended in order to evaluate one aspect of quality in an object-oriented model. The principal eigenvectors of matrices derived from the adjacency matrix of a modified class diagram are used to identify and quantify heavily loaded portions of an object-oriented design that deviate from the principle of distributed responsibilities.

Index Terms—Software metrics, object-oriented design, link analysis.

1 Introduction

THE merits of object-oriented (OO) systems concerning ease of reuse, maintainability, extensibility, and scalability are well understood and drive the wide acceptance of object-orientation among software developers. However, object-oriented design is rather a skill than a set of strict guidelines that can be safely applied. Obviously, not all object-oriented designs are of good quality. For example, novices in OO-programming or programmers with a large experience on procedural languages (who naturally find it difficult to adopt the object-oriented way of thinking [1]) tend to capture most of the domain and application semantics within a small subset of classes, occasionally within a single object [2]. In other words, the outcome of the analysis is one or more "God" objects [3], which perform most of the work in the system. Clearly, such a solution is not managing complexity any better than procedural programming.

Software metrics have been developed for evaluating and quantifying several aspects of the software engineering process [4], including metric suites for object-oriented systems [5], [6], [7], [8]. Such metrics evaluate the degree of object-orientation or measure specific characteristics of the design, such as cohesion and coupling. To the same end, metrics for evaluating the quality of a model with respect to its accordance to well-defined criteria, would be useful in object-oriented analysis [9]. Recently, empirical models for assessing high-level design quality attributes of object-oriented systems, such as reusability, flexibility, and complexity have been proposed [10].

The Hyperlink Induced Topic Search (HITS) algorithm [11] has been proposed for identifying pages on the World Wide Web that are "authoritative sources" on broad search topics. The rationale behind this algorithm is that the quality of a page p, referred to as the *authority* of the corresponding document, is not related only to the number of pages pointing to p, called *hubs*, but also to the quality of these hubs. Hubs and authorities exhibit what could be called a mutually reinforcing relationship.

This paper proposes the application of the HITS algorithm in object-oriented designs in order to evaluate the quality of a model, depicted in a class diagram or any other kind of diagram. By modifying the algorithm in order to account for the number of discrete messages exchanged between classes, it is possible to identify "God" classes [3], elements which imply a poorly designed model. The main argument is that a class cannot be considered to play a central role in a model solely on the basis of messages that it sends or receives. Whether a class is a central behavioral or data storage "God" class [3] should be determined by taking into account the importance of the classes to which it is associated into the system. Although existing OO metrics are very good at determining structural characteristics, they are not sufficient for assessing the role of each class according to the importance of its associated classes.

The rest of the paper is organized as follows: Section 2 describes briefly the required features of a quality metric concerning the identification of heavily loaded classes. In Section 3, the application of the HITS algorithm on class structures is presented along with a brief overview of the underlying mathematics and the proposed measures are introduced. Results from the application of the proposed methodology to example designs are discussed in Section 4, while existing OO metrics are discussed in Section 5. Finally, we conclude in Section 6.

2 Quality Assessment

The analysis of object-oriented systems in this study is based on the following hypothesis (the terminology is borrowed from the Web domain):

A class c holds a central role in a model:

1. *if it sends many messages to other classes which are also central or*
2. *if it receives many messages from other central classes.*

In case 1, class c is a candidate of a good hub, indicating that it sends out many requests for services to other classes that are significant to the model. In case 2, class c is a candidate of a good authority, indicating that it receives requests for services from other classes that are also of primary importance to the model. To break this circularity in the definition of good authority and hub classes, the HITS algorithm is employed.

The initial motivation behind the development of the HITS algorithm was the lack of an objective function that would be both concretely defined and corresponded to human notions of quality [11]. The goal in this paper is to define, by algorithmic means, a novel type of quality measure for the relative importance of each class in an object-oriented model.

In a balanced object-oriented design, one would expect that the responsibilities be distributed in a relatively uniform fashion among all classes of the system. Consequently, a suitable metric for evaluating whether functionality is spread uniformly would be the standard deviation of the authority and hub weights of all classes.

In the next section, the link analysis method proposed by Kleinberg [11] is modified for extracting the authority and hub weights of an object-oriented design, expressed as a modified class diagram on which the number of exchanged messages is indicated.

3 Link Analysis for OO Designs

Given a set T of associated classes, the aim is to find the authority and hub weights (a_p, h_p) associated with each class in the set. Classes with higher authority and hub weights are viewed as classes having a more important role in the model.

The set of all classes in the model can be represented as a directed graph $G = (V, E)$, where vertices correspond to the classes and a directed edge $(p, q) \in E$ indicates an association between classes p and q, with a direction from p to q. In the proposed approach, each edge is annotated with an integer $m_{p,q}$ corresponding to the number of discrete messages sent to the same direction from p to q. This slight modification to the HITS

• *The author is with the Department of Applied Informatics, University of Macedonia, 156 Egnatia st., 54006 Thessaloniki, Greece. E-mail: achat@uom.gr.*

Manuscript received 30 Dec. 2002; revised 23 Apr. 2003; accepted 6 Aug. 2003.
Recommended for acceptance by J. Offutt.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 118054.

algorithm is required since only the presence of an association is not sufficient to indicate the amount of information flow and, thus, it cannot be used to determine the relative importance of the associated classes. The algorithm starts by setting all elements of the a and h vectors equal to one.

If a class p sends many messages to classes with large a-values, it should receive a large h-value; if p receives messages from many classes with large h-values, it should receive a large a-value. By modifying the approach by Kleinberg [11], this mutually reinforcing relationship motivates the definition of the following two operations:

$$\text{Operation } I: \qquad a_p = \sum_{q:(q,p) \in E} m_{q,p} h_q, \tag{1}$$

$$\text{Operation } O: \qquad h_p = \sum_{q:(p,q) \in E} m_{p,q} a_q. \tag{2}$$

To reach an "equilibrium" for the values of a_p and h_p, the two operations can be iteratively applied in an alternating fashion until a fixed point is reached. To ensure that the algorithm converges, at each iteration, the values of a and h vectors should be normalized so that the vector has unit length [12].

From the power method of Linear Algebra [12], [13], it follows that, for a symmetric matrix A, if x is any vector not orthogonal to the principal eigenvector of A, $A^m x$ as m increases approaches the principal eigenvector of A (which in turn is the eigenvector associated with the largest eigenvalue of A).

Now, let A denote the *adjacency matrix* of the graph G in the model under study. The (i,j)th entry in A is equal to the number of messages on edge (p_i, p_j) if this edge exists; otherwise, it is equal to 0. It can be verified easily that vectors a_n and h_n are the unit vectors in the direction of $\left(A^T A\right)^{n-1} A^T z$ and $\left(AA^T\right)^n z$, respectively, where z is the vector $[1, 1, \ldots, 1]$ by which both a and h have been initialized.

Since, in the general case, $A^T z$ and z are not orthogonal to the principal eigenvectors of $A^T A$ and AA^T, respectively, the sequences $\{a_n\}$ and $\{h_n\}$ converge to the principal eigenvectors of $A^T A$ and AA^T, respectively. Since both matrices are symmetric and have real elements, the corresponding eigenvectors also have real elements [13].

Thus, to obtain the authority and hub weights of all classes, the adjacency matrix A of the graph G corresponding to the class diagram has to be found and then the authority and hub vectors are given by the normalized principal eigenvector of $A^T A$ and AA^T, respectively.

It should be noted that the power method has only been employed to prove that, if operations I and O are applied iteratively, vectors a and h will converge to the principal eigenvectors of $A^T A$ and AA^T, respectively. The actual computation of the eigenvectors can be performed using more efficient algorithms, such as Schur decomposition [13].

To determine whether a class plays the role of a "God" object within the system, both its authority and its hub weight have to be examined. In case a class acts as a behavioral "God" class [3], initiating requests for services to the rest of the system, it will obtain a high hub value, while, in case it acts as a Data Structure "God" class receiving messages, it will obtain a high authority value. Thus, a good measure for this purpose would be the average of these two weights.

Since the quality of a design cannot be determined solely on the basis of either the authority of the hub weights, we define as responsibility *distribution quality* metric of a class-based system represented by a graph G, the standard deviation of the elements in a vector containing the average of the authority and hub weights assigned to each of the classes in the system:

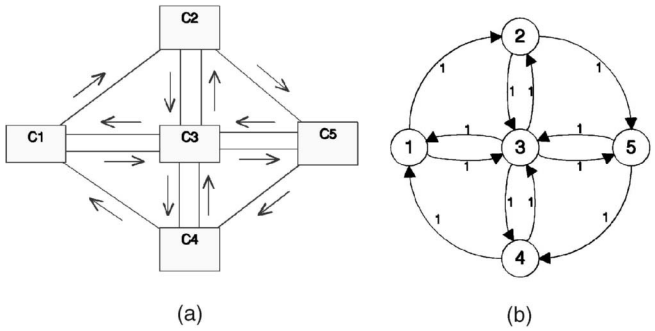

Fig. 1. (a) OO design with a "God" class and (b) corresponding annotated graph.

$$dq_G = \sigma\left(\frac{a_n + h_n}{2}\right). \tag{3}$$

A low standard deviation indicates more uniformity and therefore more conformance with the distributed responsibility principle.

So far, the proposed analysis has not considered inheritance, which is a vital characteristic of many OO designs. In case of a generalization relationship, the derived classes inherit all attributes and methods from their base class. Consequently, derived classes will also inherit associations from their ancestors, which are the "information pipelines" through which they can exchange messages. However, since each descendant can either inherit the base class methods without modification, change their implementation either through static binding (overriding) or dynamic binding (polymorphism), or even add new methods, the number of exchanged messages in the adjacency matrix should be extracted for each class separately.

Similar to the *Coupling between objects* (CBO) metric [5], inheritance related classes should not be considered as being coupled and, therefore, they should not contribute to the "authority" of "hub" weights of the associated classes. Only explicit exchange of messages should be taken into account. The reason why link analysis cannot take advantage of the inheritance relationship to reduce the amount of computation is the following: Although, according to the Liskov Substitution Principle [14], the subclass must be substitutable for its base class and, consequently, should be able to receive the same messages; any derived class might also have additional associations and, thus, send/receive messages to/from other classes as well, which is apparent in several Design Patterns [15].

4 Examples

To illustrate the necessity for computing the authority/hub weights in order to estimate weaknesses in the design of an object-oriented system, let us consider the following example, where a class c acts as a controller. In the diagram of Fig. 1a, class C3 is actually a central "brain" class controlling behavior and initiating any activity in the system [3]. The notation of collaboration diagrams in UML is employed to show the number of discrete messages exchanged between classes.

From the graph corresponding to this diagram (Fig. 1b), the adjacency matrix that is obtained is:

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

The principal eigenvectors for the $A^T A$ and AA^T matrices, corresponding to the principal eigenvalues, are:

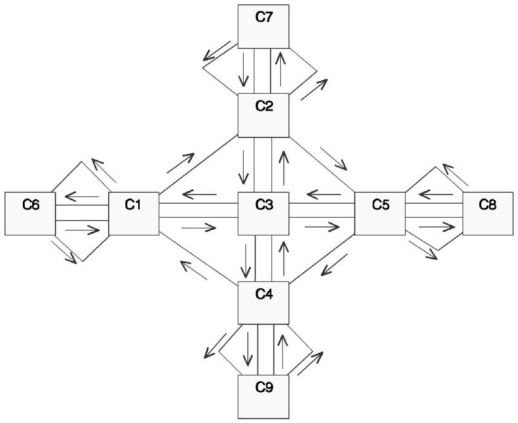

Fig. 2. OO design with "God" class and helper classes.

$$a_n = \begin{bmatrix} 0.394 \\ 0.394 \\ 0.615 \\ 0.394 \\ 0.394 \end{bmatrix}, \qquad h_n = \begin{bmatrix} 0.394 \\ 0.394 \\ 0.615 \\ 0.394 \\ 0.394 \end{bmatrix}.$$

The elements of these vectors are the authority and hub weights of the corresponding nodes/classes according to the application of the modified HITS algorithm. From the authority and hub weights of class *C*3, it becomes obvious that this class has a central role in the system, encompassing most of the system's intelligence and, thus, violating the principle of uniformly distributed responsibilities. Such a class corresponds directly to the role of a controller function in procedural programming. Due to the symmetry of the diagram, all other classes have equal weights and also vectors a_n and h_n are equal.

The reason for choosing authority/hub weights instead of other measures, such as the number of incoming/outgoing messages (in or out-degree) for each class (which would also result in the same ordering in this case), is best explained through a second example. Suppose that each of the peripheral classes in the previous example (*C*1, *C*2, *C*4, and *C*5) had a helper class for performing secondary functions (Fig. 2). It is clear that *C*3 remains the central "brain" class of the system. If the number of incoming and outgoing messages is used for identifying central classes in the design, classes *C*1, *C*2, *C*4, and *C*5 cannot be distinguished from class *C*3 (since all have an in and out-degree of 4). On the other hand, the calculation of the authority/hub weights by means of the adjacency matrix (where edges corresponding to associations on which two messages are exchanged have a weight of two) results in:

$$a_n^T = h_n^T = [0.383 \quad 0.383 \quad 0.454 \quad 0.383 \quad 0.383 \quad 0.227 \quad 0.227 \quad 0.227 \quad 0.227].$$

Class *C*3 is clearly identified by the authority/hub weights as the most heavily loaded class in the design. Helper classes are associated to the lowest authority/hub weights.

To evaluate the proposed method in a nonideal example, we consider an object-oriented system for modeling the operation of a microwave oven, developed both in a "novice design" manner in which a central "Manager"-like object captures most of the functionality and in a more sophisticated way in which the "God" object has been eliminated [2]. A diagram showing the classes of the initial design and the messages exchanged is shown in Fig. 3. For this system, the authority and hub weights are given by the vectors:

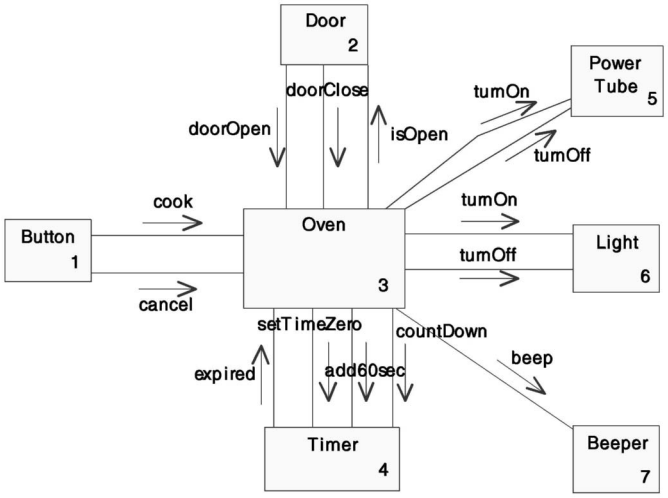

Fig. 3. Modified collaboration diagram for oven system.

$$a_n^T = [0 \quad 0.229 \quad 0 \quad 0.688 \quad 0.459 \quad 0.459 \quad 0.229]$$
$$h_n^T = [0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0].$$

It should be mentioned that the central class has an authority weight of zero since classes sending messages to it do not receive messages by any other class than the central. This has been called the *nil-weighting* limitation of the HITS algorithm [16]. However, it has a hub weight of one, indicating clearly that this class initiates any activity in the system.

In the improved design shown in Fig. 4, the oven class has been eliminated in recognition that there is no need to have major control objects between the generator and the processor of an event. For this system, which emphasizes the principles of encapsulation and delegation, the corresponding vectors are:

$$a_n^T = [0.428 \quad 0.433 \quad 0.219 \quad 0.759 \quad 0 \quad 0.053]$$
$$h_n^T = [0 \quad 0 \quad 0.776 \quad 0 \quad 0.195 \quad 0.6].$$

In this design, authority and hub weights are much more uniformly distributed. The distribution quality metric for the initial and the improved oven system has a value of 0.165 and 0.141, respectively.

5 Related Work

The literature review on software metrics reveals a plethora of measures for a multitude of aspects of OO designs. To validate the use of the proposed metric, well-established internal product metrics from the suites proposed by Chidamber and Kemerer [5], Lorenz and Kidd [6], Brito e Abreu [7], and Li and Henry [8] have been selected for comparison.

The proposed link-analysis metric focuses mainly on a specific design heuristic, which states that the designer should avoid creating "God" classes/objects in the system [3], [9]. It aims first to identify such classes in the system and, second, to evaluate the degree of responsibility distribution among the classes of a design. Consequently, only metrics that can be directly or indirectly employed for this purpose have been considered. The comparison should therefore be made against the following criteria:

1. The ability to account for the significance of the related classes (whether, for example, the metric can differentiate

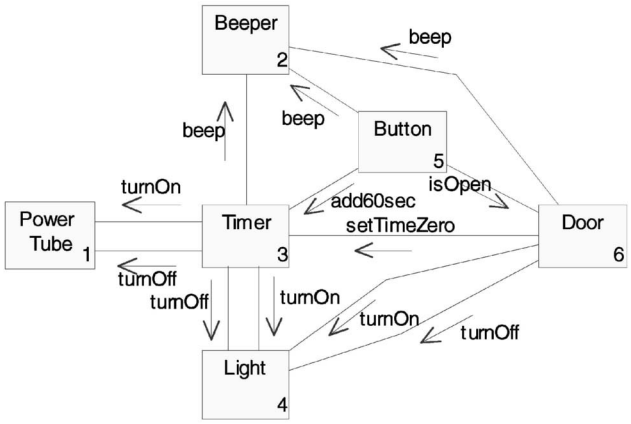

Fig. 4. Improved oven system design.

between class $C3$ and peripheral classes $C1$, $C2$, $C4$, and $C5$ in Fig. 2).

2. The ability to consider both incoming and outgoing flows of messages (for example, if one class $C1$ sends out n messages, while another class $C2$ receives *and* sends n messages, the metric should differentiate between the roles of the two classes).

One could argue that classes with a high static complexity, which is captured by the *Weighted Methods per Class* (WMC) metric [5], are possibly "God" classes. However, a large WMC value, which could also be the result of a single overcomplicated function, does not imply that the class has a central role in the system in the sense that many other central classes are using it or providing services to it. Moreover, this metric is not applicable at any phase prior to detailed design.

In a similar manner, since the *Lack of Cohesion in Methods* (LCOM) metric [5] captures the degree of cohesiveness of methods within a class, it could be used for determining "God" classes, when disjoint functionalities have been placed into one class, which should then probably be split into two or more classes. This metric, however, does not guarantee the number and significance of collaborators (i.e., a class with a high LCOM value could possibly be used by a single other class). For both the WMC and LCOM metrics, criteria 1 and 2 are not fulfilled. On the other hand, these metrics could be used complementary to the proposed one in order to differentiate between classes which appear to be equally important to the system according to link analysis (for example, peripheral classes $C1$, $C2$, $C4$, and $C5$ in Fig. 2).

From the metrics suite proposed by Lorenz and Kidd [6], the *Number of message sends* (NOM) summed over all class methods and the *Number of Instance Methods in a Class* (NIM) (assuming that the latter is related to the number of received messages) can also be considered an alternative. According to Lorenz and Kidd, a high NIM value indicates a large class that may be trying to do too much of the work itself instead of putting the responsibilities where they belong. However, these metrics end up in the calculation of in/out degree, which, as already explained, is not sufficient to distinguish between central and peripheral classes as in the example of Fig. 2, although it fulfills criterion 2. Metrics which have a similar motivation to the proposed one are the *Number of Key Classes* (NKC) and the *Number of Support Classes* (NSC) [6], where key classes are those focused on the application domain and appear to have a central role, while support classes tend to be more application-specific. However, the identification of key classes lacks formality and, since it takes place during analysis, one key class might loose this attribute in subsequent phases.

The degree of coupling measured by either the *Coupling between objects* (CBO) [5] at a class level or the *Coupling Factor* (CF) from the MOOD set of metrics [7] at a system level, providing insight to the "fan-out" of each class, could signify that the class with the highest coupling factor is a "God" class. To the same end, the *Message-Passing Coupling* (MPC) [8], defined as the sum of the number of method calls made by all methods in a class, could also be used. With all these metrics it is possible to quantify a class's complexity. Classes with high CBO and MPC metrics may be doing too much work and should be split into smaller, more narrowly focused classes. However, these metrics do neither take into account the significance of the related classes nor the number of classes referencing the class and, thus, they fail to account for "authority/hub" weights in the system.

6 Conclusions

A method for assessing the design quality of an object-oriented model in terms of responsibility distribution among the classes of the system has been proposed. The role of each class in a class diagram depends not only on the number of incoming and outgoing messages, but also on the importance of the classes to which it is associated. A link analysis method currently employed for information retrieval from the Web has been extended for obtaining authority and hub weights for each class, capturing the combined effect of communicating with other classes for servicing or issuing requests. These weights are obtained employing the adjacency matrix of the directed graph that corresponds to the class diagram, annotated with the number of messages being exchanged. Finally, the standard deviation of these weights is proposed as a measure of the uniform distribution of responsibilities in a model.

References

[1] T. Budd, *An Introduction to Object-Oriented Programming.* Addison-Wesley, 2001.
[2] R.C. Lee and W.M. Tepfenhart, *UML and C++: A Practical Guide To Object-Oriented Development.* Prentice Hall, 2001.
[3] A.J. Riel, *Object-Oriented Design Heuristics.* Addison-Wesley, 1996.
[4] N.E. Fenton and S.L. Pfleeger, *Software Metrics: A Rigorous & Practical Approach.* Int'l Thompson Publishing, 1997.
[5] S.R. Chidamber and C.F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Trans. Software Eng.*, vol. 20, no. 6, pp. 476-493, June 1994.
[6] M. Lorenz and J. Kidd, *Object-Oriented Software Metrics.* Object-Oriented Series, Prentice Hall, 1994.
[7] F. Brito e Abreu, "The MOOD Metrics Set," *Proc. Ninth European Conf. Object-Oriented Programming (ECOOP '95) Workshop Metrics,* Aug. 1995.
[8] W. Li and S. Henry, "Object-Oriented Metrics that Predict Maintainability," *J. Systems and Software,* vol. 23, no. 2, pp. 111-122, Nov. 1993.
[9] C. Kirsopp, M. Shepperd, and S. Webster, "An Empirical Study into the Use of Measurement to Support OO Design Evaluation," *Proc. Sixth IEEE Int'l Symp. Software Metrics,* pp. 230-241, Nov. 1999.
[10] J. Bansiya and C.G. Davis, "A Hierarchical Model for Object-Oriented Design Quality Assessment," *IEEE Trans. Software Eng.*, vol. 28, no. 1, pp. 4-17, Jan. 2002.
[11] J.M. Kleinberg, "Authoritative Sources in a Hyperlinked Environment," *J. ACM,* vol. 46, no. 5, pp. 604-632, Sept. 1999.
[12] C.D. Meyer, *Matrix Analysis and Applied Linear Algebra.* Philadelphia: SIAM, 2000.
[13] G. Golub and C.F. Van Loan, *Matrix Computations.* Johns Hopkins Univ. Press, 1996.
[14] R.C. Martin, *Agile Software Development: Principles, Patterns and Practices.* Prentice Hall, 2003.
[15] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.
[16] J.C. Miller, G. Rae, and F. Schaefer, "Modifications of Kleinberg's HITS Algorithm Using Matrix Exponentiation and Web Log Records," *Proc. 24th Int'l ACM SIGIR Conf. Research and Development in Information Retrieval,* Sept. 2001.