Evaluating Object-Oriented Designs with Link Analysis

Alexander Chatzigeorgiou, Spiros Xanthos and George Stephanides
Department of Applied Informatics, University of Macedonia
54006 Thessaloniki, Greece
E-mail: {achat, it0187, steph}@uom.gr

Abstract

The Hyperlink Induced Topic Search algorithm, which is a method of link analysis, primarily developed for retrieving information from the Web, is extended in this paper, in order to evaluate one aspect of quality in an object-oriented model. Considering the number of discrete messages exchanged between classes, it is possible to identify "God" classes in the system, elements which imply a poorly designed 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. The non-principal eigenvectors are also employed in order to identify possible reusable components in the system. The methodology can be easily automated as illustrated by a Java program that has been developed for this purpose.

1. Introduction

The object-oriented (OO) paradigm promises to be one of the most flexible frameworks for developing systems by shifting responsibility from functional modules to a more local level. The merits of object-oriented systems concerning ease of reuse, maintenance, 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 [2]), tend to capture most of the domain and application semantics within a small subset of classes, occasionally within a single object [13]. In other words, the outcome of the analysis is often one or more "God" classes [20], which perform most of the work in the system. Clearly, such a solution is not managing complexity any better than procedural programming.

The literature review on software metrics reveals a large number of methods that have been developed for evaluating and quantifying aspects of the software engineering process [4], among them metric suites for object-oriented systems [3]. Such metrics help to evaluate the degree of object-orientation or measure specific characteristics of the design, such as cohesion and coupling. However, there is a lack of metrics that can evaluate the conformance of an object-oriented model to well established design heuristics [11].

The abundance problem in broad-topic queries on the World Wide Web has triggered several research efforts aiming at reducing the set of returned pages to the most "definite" ones. The Hyperlink Induced Topic Search (HITS) algorithm [12] is a theoretically justified approach 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 a modified HITS algorithm in object-oriented designs, in order to evaluate the quality of a model, depicted in a class or collaboration 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 [20], elements that 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 [20] should be determined by taking into account the importance of the classes to which it is associated, into the system.

In addition, the algorithm is also used in order to identify dense communities of classes in the system which are well-separated from one another. Such dense

Proceedings of the 26th International Conference on Software Engineering (ICSE'04)
0270-5257/04 $20.00 © 2004 **IEEE**

communities might indicate groups of classes with a distinct role and which are loosely connected to the rest of the system, thus implying possible reusable components. The proposed methodology can easily be automated: A Java based program that has been implemented illustrates visually both the workload of each class as well as the identified possible reusable portions of the design.

Another method inspired by Web document ranking, namely the Component Rank, has been utilized for ranking software components based on analyzing use relations [10]. However, this approach does not aim to identify weaknesses in a system's design but rather to obtain generic components. Moreover, it depends on empirically amended parameters, such as the values of distribution ratios, ratio between real and pseudo use relations and similarity thresholds.

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 an overview of the underlying mathematics, and the proposed measures are introduced. Results from the application of the proposed methodology to example designs are given in section 4, while an outline of the developed program is presented in section 5. Several related metrics are discussed in section 6. Finally, we conclude in section 7.

2. Quality Assessment

Paraphrasing Kleinberg's claim [12], the link structure of an object-oriented design can be a rich source of information regarding the content of the environment, provided that we have effective means for understanding it. 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:

a. if it receives many messages from other central classes.

b. if it sends many messages to other classes which are also central

In case (a), 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. In case of (b), 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. Obviously, the above definition of good authority and hub classes is based on an inherent circularity. To break this circularity, 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 [12]. 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.

For a well-designed object-oriented system, one would expect that the responsibilities be distributed in a relatively uniform fashion among all classes of the system. This is in accordance to the Single-Responsibility Principle (SRP) [17], which states that *a class should have only one reason to change*. Expressed in terms of cohesion, a class should not aggregate separate responsibilities becoming a class with a large set of signatures. Such a design would probably be much more flexible to changes without causing degradation to the initial design. 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.

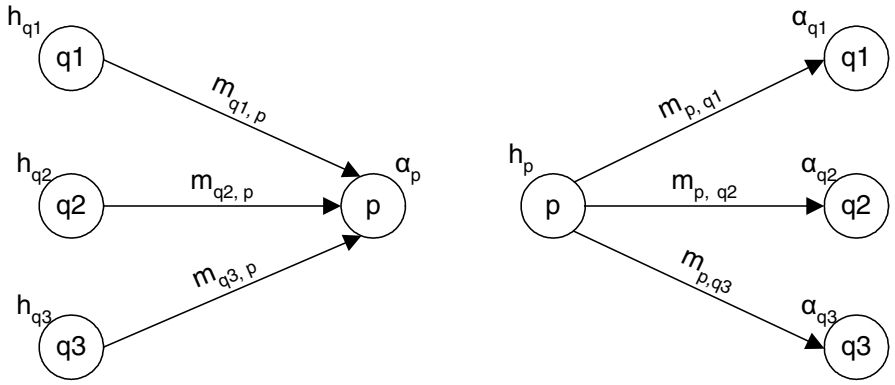
In the next section, the link analysis method proposed in [12] is modified for extracting the authority and hub weights of an object-oriented design, expressed as a modified class diagram on which the number of exchanged messages is indicated. Moreover, the possibility to employ the algorithm in order to identify distinct communities of classes is discussed.

3. Link Analysis Method

3.1 Identification of "God" classes

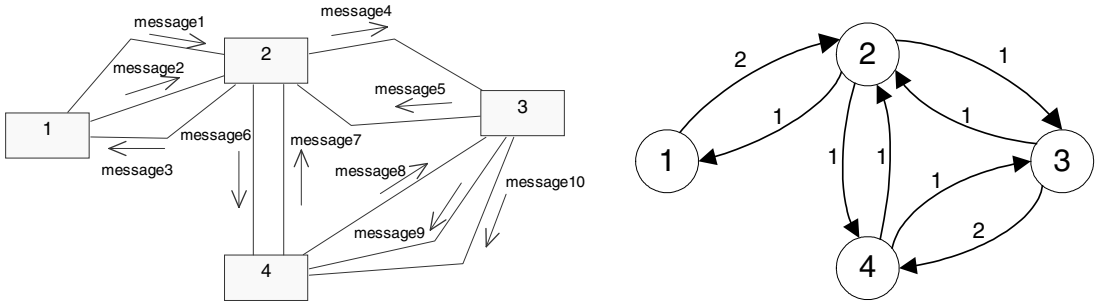
Given a set T of associated classes, the aim is to find the authority and hub weights (a_p, h_p), associated with each class in the set. Classes with higher authority and hub weights are viewed as classes having a more important role in the model. The set of all classes in the model can be represented 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. 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 in [12], this mutually reinforcing relationship motivates the definition of the following two operations:



Operation *I* **Operation *O***

Figure 1: Definition of authorities and hubs



(a) Class Diagram (b) Corresponding graph

Figure 2: Hypothetical object-oriented design

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

Operation *O*: $$h_p = \sum_{q:(p,q)\in E} m_{p,q} a_q \quad (2)$$

Both operations are graphically depicted in Figure 1. 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 [18].

As an example, the hypothetical object-oriented design shown in Figure 2(a) can be represented by the weighted directed graph in Figure 2(b). The system of two sets of algebraic equations that gives the authority and hub weights for each of the nodes (classes) is:

$$\begin{aligned} a_1 &= 0\cdot h_1 + 1\cdot h_2 + 0\cdot h_3 + 0\cdot h_4 \\ a_2 &= 2\cdot h_1 + 0\cdot h_2 + 1\cdot h_3 + 1\cdot h_4 \\ a_3 &= 0\cdot h_1 + 1\cdot h_2 + 0\cdot h_3 + 1\cdot h_4 \\ a_4 &= 0\cdot h_1 + 1\cdot h_2 + 2\cdot h_3 + 0\cdot h_4 \end{aligned} \quad , \quad \begin{aligned} h_1 &= 0\cdot a_1 + 2\cdot a_2 + 0\cdot a_3 + 0\cdot a_4 \\ h_2 &= 1\cdot a_1 + 0\cdot a_2 + 1\cdot a_3 + 1\cdot a_4 \\ h_3 &= 0\cdot a_1 + 1\cdot a_2 + 0\cdot a_3 + 2\cdot a_4 \\ h_4 &= 0\cdot a_1 + 1\cdot a_2 + 1\cdot a_3 + 0\cdot a_4 \end{aligned}$$

which can be written as:

$$a = A^T h\,, \quad h = A a$$

where a, h are the vectors of the authority and hub weights, respectively, and A denotes the *adjacency matrix* of the graph G in the model under study. The $(i, j)^{\text{th}}$ 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:

$$A = \begin{bmatrix} 0 & 2 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

The above system can be solved iteratively (Gauss-Seidel method), using as an initial guess the unit vector. The solution converges after a limited number of iterations if vectors a, h are normalized at each iteration.

From the power method of Linear Algebra [6, 18] it follows that for a symmetric matrix A, if x is any vector not orthogonal to the principal eigenvector of A (i.e. the eigenvector associated with the largest eigenvalue of A),

$A^m x$ approaches the principal eigenvector of A as m increases.

Since the I and O operations can be written as $a \leftarrow A^T h$ and $h \leftarrow Aa$, respectively, it can be verified easily that vectors a_n and h_n (which are vectors a and h during the n-th iteration) 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, … ,1] by which both a and h have been initialized.

For our example, matrices A^T and A are the matrices of the coefficients for the two sets of equations. The solution for the α vector at iteration n (denoted as a_n) is given by:

$$a_n = A^T h_{n-1} = A^T \left(A a_{n-1}\right) = A^T \left(A\left(A^T h_{n-2}\right)\right) =$$
$$\ldots = \left(A^T A\right)^{n-1} A^T z$$

and in a similar manner

$$h_n = A a_n = A \left(A^T h_{n-1}\right) = \ldots = \left(A A^T\right)^n z$$

According to *Perron's Theorem* [16] the eigenvalue of largest absolute value of a positive (square) matrix is positive and belongs to a positive eigenvector (i.e. the principal eigenvector). As a result, since matrices $A^T A$ and AA^T are symmetric and have only positive entries, their principal eigenvectors are also positive. Thus, vectors $A^T z$ and z (which are positive) are not orthogonal to the principal eigenvectors of $A^T A$ and AA^T, respectively, since for two vectors to be orthogonal their dot product (which is the sum of products of their entries) should be zero. Consequently, according to the Power Method stated earlier, the sequences $\{a_n\}$ and $\{h_n\}$ converge to the principal eigenvectors of $A^T A$ and AA^T, respectively.

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.

To determine whether a class plays the role of a "God" class within the system, both its authority and its hub weight have to be examined. In case a class acts as a behavioral "God" class [20] 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.

According to the above, the quality of a design cannot be determined solely on the basis of either the authority of the hub weights. Therefore, 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:

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

The lower the standard deviation, the more the model conforms to the principle of distributed responsibilities among the classes of the system. It should be mentioned that the standard deviation is the most commonly used measure of how spread out a distribution is. Consequently, it serves perfectly the purpose of identifying how spread out the responsibilities in a community of interacting objects is.

The above process can be easily automated to obtain the authority and hub weights of all classes, given the class diagram and the corresponding sequence diagrams from which the number of exchanged messages can be found. Both kinds of diagrams are commonplace in most software engineering tools supporting UML notation.

3.2 Identification of reusable components

The analysis of the link structure of an object-oriented design by means of eigenvectors can be further exploited in order to identify strongly coupled groups of classes in the system. The idea comes from the identification of multiple communities (regarding a query topic) in a hypermedia environment proposed in [12]. The motivation here is to find dense communities of classes (i.e. classes which exchange a large number of messages), which are well separated from one another.

The presence of a dense community of classes that is loosely coupled to other classes or communities, might imply relevance of functionality between those classes and thus indicate a portion of the overall design that has a distinct purpose. Such an identification of dense communities (if they exist) can serve as an indicator of reusable components in the system that can be ported to other settings. Finding such groups of classes by automated means can be valuable in maintaining and enhancing the system.

In order to identify dense communities of classes the non-principal eigenvectors of the matrices $A^T A$ and AA^T can be used. These non-principal eigenvectors, whose elements are the authority and hub weights of the corresponding classes, will have both positive and negative entries. Each pair of eigenvectors (a_i^*, h_i^*) associated with the same eigenvalue will therefore provide two communities of authorities and hubs. One consisting of the classes with the most positive

coordinates in a_i^* and h_i^* and one consisting of the classes with the most negative coordinates. Each set of

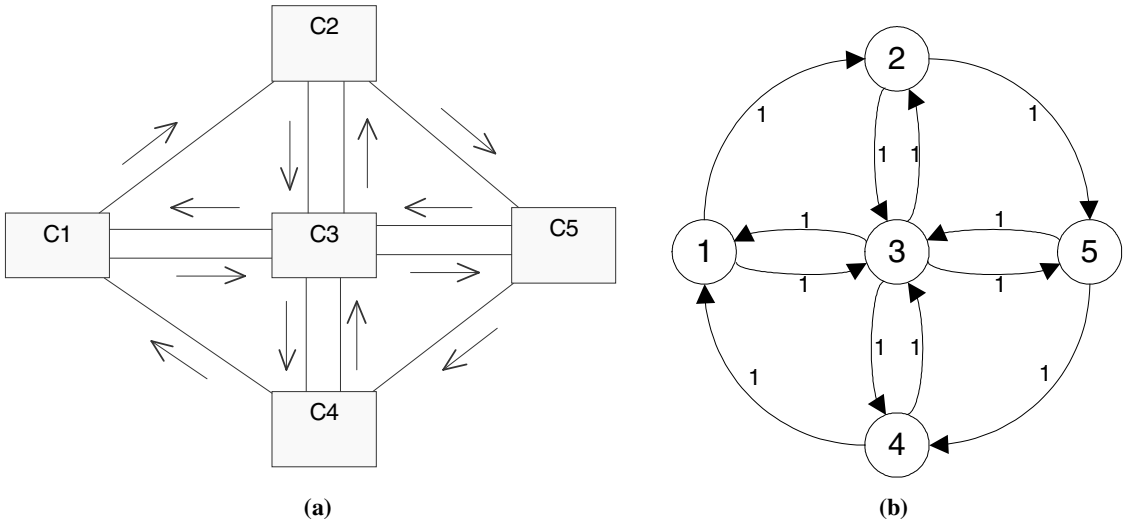


Figure 3: (a) OO Design with a "God" class and (b) corresponding annotated graph

coordinates corresponds to classes that are densely coupled, while the two communities are expected to be very sparsely connected in the underlying graph representing the system [12]. It should be noted that the eigenvalue to which the non-principal eigenvectors are associated, indicates the strength of the community, i.e. the extent to which hub and authority weights reinforce each other.

4. Application results

The necessity for computing the authority/hub weights, in order to estimate weaknesses in the design of an object-oriented system, will be illustrated through the following example, where one class acts as a controller. In the diagram of Figure 3(a), class *C*3 is actually a central brain class controlling behavior and initiating any activity in the system [20].

From the graph corresponding to this diagram (Figure 3(b)), the adjacency matrix that is obtained is:

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

(all non-zero elements are equal to 1, since each class exchanges in this example at most one message with other classes).

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

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

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.

One reasonable question is why authority/hub weights should be used 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). The answer 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 (Figure 4). 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, *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

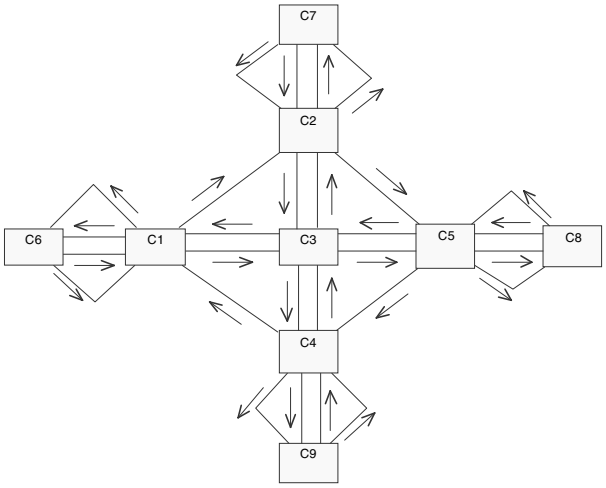


Figure 4: OO Design with "God" class and helper classes

(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 non-ideal 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" class object has been eliminated [13]. A diagram showing the classes of the initial design and the messages exchanged is shown in Figure 5. For this system the authority and hub weights are given by the vectors:

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

As it can be observed 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 in [19]. 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 Figure 6, 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

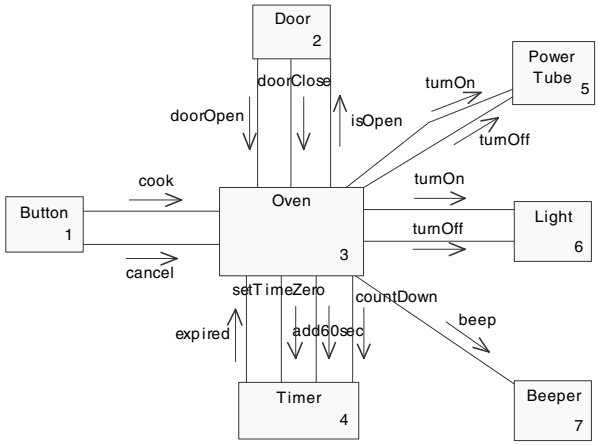


Figure 5: Initial design for oven system

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, implying a more balanced system. The distribution quality metric for the initial and the improved oven system has a value of 0.165 and 0.141, respectively.

In this design, although the "God" class has been eliminated, the coupling between classes has increased significantly. However, the new design conforms to the Single Responsibility Principle much better that the initial one. A further refinement to the design can be achieved by resolving the high coupling problem using Design Patterns, namely the *Observer* and the *Adapter* pattern [5, 13].

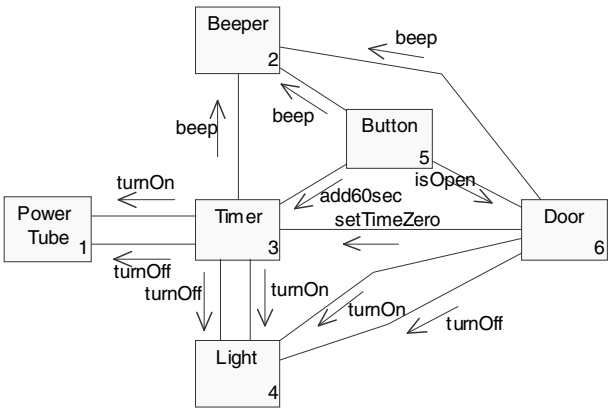


Figure 6: Improved oven system design

Since there are design principles such as the *Interface-Segregation Principle* [17] and design patterns such as the *Adapter Pattern* [5] that handle the problem of classes with "fat" interfaces, the proposed metric should be further evaluated against Design Patterns. If a Design Pattern is supposed to improve the quality of an object-oriented design, such a qualitative improvement should be measurable by the applied metrics. Any of the metrics from the plethora of suggested ones that does not validate the usefulness of well established Design Patterns should possibly be discarded.

The proposed metric so far does not ensure a proper handling of polymorphic behavior, since it is based on explicit message counting. In order to calculate in a fairer manner the authority and hub weights of classes that may be bound at runtime, one might have to employ pseudonodes and/or to divide the weight of an edge by the number of all possible message receivers. This is particularly important when applying the proposed metric on systems that employ design patterns, since almost all patterns are based on polymorphism.

The use of non-principal eigenvectors in order to identify dense communities of classes within an object-oriented system will be shown with a simple system, inspired by the Observer design pattern [5]. In this example the separation is particularly striking; however, it should be made clear that finding such communities is not always trivial. In many cases the methodology might not be helpful at all, since it requires a relatively clear separation of the classes. If such a separation does not exist, positive and negative entries in the corresponding eigenvectors might not be particularly distinct. On the other hand, such a situation possibly implies a system where all classes are more or less coupled leaving no room for identifying reusable components.

The system under study is shown in Figure 7. As it can be visually observed, the system is supposed to have three groups of classes: one group is formed around the subject of the Observer pattern and the two others correspond to the observers and their communicating classes. The adjacency matrix in this case is a 9×9 matrix.

The first non-principal eigenvectors of the $A^T A$ and AA^T matrices are:

$$a^T = [0.102\ 0.048\ 0.074\ -0.359\ -0.143\ -0.541\ 0.519\ 0.499\ 0.138]$$
$$h^T = [0.053\ 0.122\ 0.051\ -0.313\ -0.577\ -0.123\ 0.171\ 0.301\ 0.641]$$

while the second non-principal eigenvectors are:

$$a^T = [-0.613\ -0.391\ -0.470\ 0.077\ 0.082\ 0.258\ 0.232\ 0.330\ 0.074]$$
$$h^T = [-0.414\ -0.646\ -0.370\ 0.167\ 0.238\ 0.028\ 0.121\ 0.150\ 0.386]$$

In the first set the negative entries clearly identify the group of classes around class 4-*Observer* (classes 4, 5, 6)

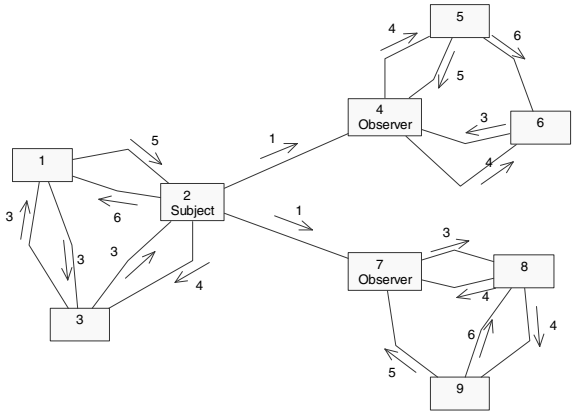


Figure 7: Sample system with three communities

emphasizing their separation from the rest of the system. The second non-principal eigenvectors imply a separation for the classes around the *Subject* class (classes 1, 2, 3). The authority and hub weights themselves have exactly the same meaning as in the principal eigenvector, i.e. they indicate the degree of their mutual reinforcing relationship.

5. Visualization

Obviously, one of the advantages in using software metrics instead of design heuristics in order to assess the quality of an object-oriented system is the possibility for automation. The application of the proposed algorithm, as implemented in a Java program is shown in Figure 8.

It is assumed that the class diagram annotated with the number of exchanged messages is described in an XML

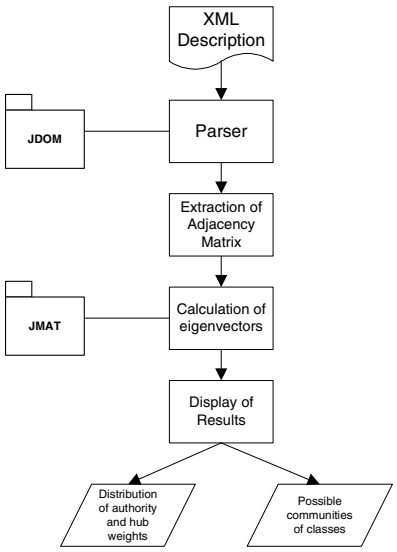


Figure 8: Program flowchart

file. The program parses the XML file with the help of the org.jdom package [9] and extracts the adjacency matrix A. It then uses the methods provided by the Java Matrix tools package [8] that provides MATLAB-like functions and syntax, in order to calculate the principal as well as the non-principal eigenvectors of the $A^T A$ and AA^T matrices. The results are then used in order to generate the images showing the distribution of authority/hub weights and the identified communities of classes.

The tool constructs separate images for the distribution of authority and hub weights. In order to color each class according to the its authority or hub weight, the red and blue colors of the RGB value scheme are employed: if one class is assigned an authority/hub weight of one, it is colored red, while in case of a zero weight, it is colored blue. Consequently, the distribution of the "workload" in the system is displayed as a distribution of colors in the red-blue spectrum and bright red points indicate heavily loaded portions of the design, while deep blue classes correspond to "light" portions.

Figure 9 shows a sample screenshot of the program, displaying the distribution of authority weights by means of colored classes, for the example of Figure 4. The classes have been placed differently on purpose, in order to emphasize the ease in recognizing the "hot" classes.

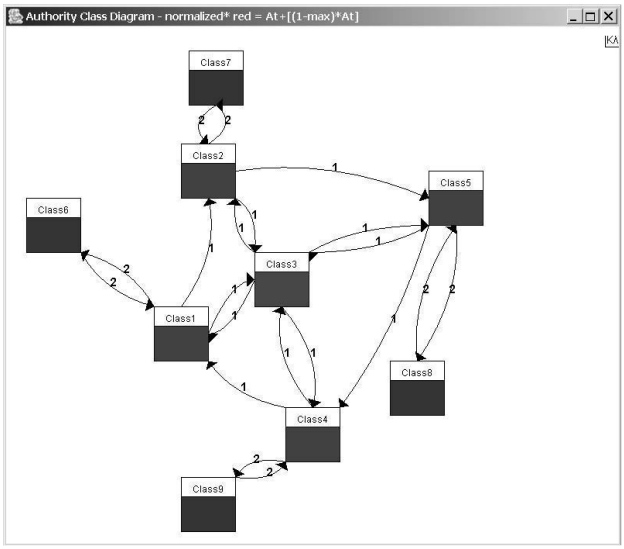


Figure 9: Distribution of authority weights for the example of Fig. 4

The program can also provide an image displaying the identified class communities, based on the information retrieved from non-principal eigenvectors. A sample screenshot for the example of Figure 7, showing encircled two of the three groups of classes is shown in Figure 10.

The source code of the Java program is available and can be downloaded from [7].

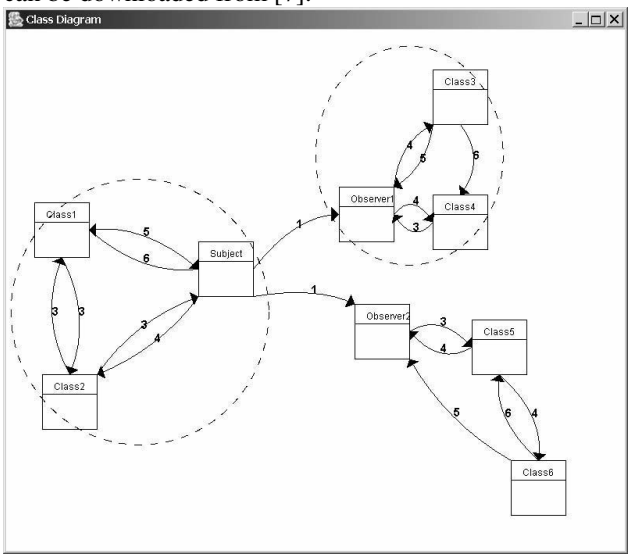


Figure 10: Identification of possible communities of classes by means of non-principal eigenvectors

6. Related Work

Several software metrics and measures can be found in the relevant literature for a multitude of aspects of OO designs. 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 [11, 20]. It aims firstly to identify such classes in the system and, secondly, to evaluate the degree of responsibility distribution among the classes of a design. Consequently, any comparison should be made against the following two criteria:

a) the ability to account for the significance of the related classes (whether for example the metric can differentiate between class *C*3 and peripheral classes *C*1, *C*2, *C*4 and *C*5 in Figure 4).

b) the ability to consider both incoming and outgoing flows of messages (for example if one class *C*1 sends out *n* messages, while another class *C*2 receives *and* sends *n* messages, the metric should differentiate between the roles of the two classes).

Metrics which have a similar motivation to the proposed one are the *Number of Key Classes* (NKC) and the *Number of Support Classes* (NSC) [15], 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.

Considering one widely accepted suite of metrics [3], the *Weighted Methods per Class* (WMC) metric can identify classes with a high static complexity. 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 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. This metric however, does not guarantee the number and significance of collaborators. For both the WMC and LCOM metrics, criteria a) and b) are not fulfilled.

From the metrics suite proposed by Lorenz and Kidd [15] 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. 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 Figure 4, although it fulfills the criterion b).

The *Coupling between objects* (CBO) [3] metric at a class level or the *Coupling Factor* (CF) from the MOOD set of metrics [1] at a system level both measure the degree of coupling, providing insight to the "fan-out" of each class. To the same end, the *Message-Passing Coupling* (MPC) [14], 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.

7. Conclusions

A method for evaluating quality in an object-oriented design in terms of responsibility distribution among the classes of the system has been proposed. The method is based on the observation that the role of each class in a system depends not only on the number of incoming and outgoing messages, but also on the importance of the classes to which it is associated. The proposed method extends a link analysis algorithm currently employed for information retrieval from the Web. Authority and hub weights are obtained for each class, capturing the combined effect of communicating with other classes for servicing or issuing requests. The algorithm can be further used in order to identify dense communities of classes in the system, which possibly implies reusable components.

8. References

[1] F. Brito e Abreu, "The MOOD Metrics Set," *Proc. 9th European Conference on Object-Oriented Programming* (*ECOOP'95*) *Workshop Metrics*, Åarhus, Denmark, Aug. 1995.

[2] T. Budd, *An Introduction to Object-Oriented Programming*, Addison-Wesley, Boston, MA, 2001.

[3] S. R. Chidamber, C.F. Kemerer, "A Metrics Suite for Object Oriented Design", *IEEE Transactions on Software Engineering*, vol. 20, no. 6, June 1994, pp. 476-493.

[4] N. E. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous & Practical Approach,* International Thompson Publishing, Boston, MA, 1997.

[5] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Boston, MA, 1995.

[6] G. Golub, C. F. Van Loan, *Matrix Computations*, Johns Hopkins Univ Pr., Baltimore, 1996.

[7] http://java.uom.gr/~spiros/linkanalysis

[8] http://sourceforge.net/projects/jmat/

[9] http://www.jdom.org

[10] K. Inoue, R. Yokomori, H. Fujiwara, T. Yamamoto, M. Matsushita, S. Kusumoto, *Proc. 25th Int. Conference on Software Engineering* (ICSE'03), Portland, Oregon, USA, May 2003.

[11] C. Kirsopp, M. Shepperd, S. Webster, "An Empirical Study into the Use of Measurement to Support OO Design Evaluation", *Proc. 6th IEEE Int. Symposium on Software Metrics*, Boca Raton, FL, USA, Nov. 1999, pp. 230-241.

[12] J. M. Kleinberg, "Authoritative Sources in a Hyperlinked Environment", *Journal of the ACM*, vol. 46, issue 5, Sep. 1999, pp. 604-632.

[13] R. C. Lee and W. M. Tepfenhart, *UML and C++: A Practical Guide To Object-Oriented Development*, Prentice Hall, Upper Saddle River, NJ, 2001.

[14] W. Li, S. Henry, "Object-Oriented Metrics that Predict Maintainability", *Journal of Systems and Software*, vol. 23, no. 2, Nov. 1993, pp. 111-122.

[15] M. Lorenz and J. Kidd, *Object-Oriented Software Metrics*, Object-Oriented Series, Prentice Hall, Englewood Cliffs, NJ, 1994.

[16] C. R. MacCluer, "The Many Proofs and Applications of Perron's Theorem", *SIAM Review*, vol. 42, no. 3, 2000, pp. 487-498.

[17] R. C. Martin, *Agile Software Development: Principles, Patterns and Practices,* Prentice Hall, Upper Saddle River, NJ, 2003.

[18] C. D. Meyer, *Matrix Analysis and Applied Linear Algebra,* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 2000.

[19] J. C. Miller, G. Rae, F. Schaefer, "Modifications of Kleinberg's HITS Algorithm using Matrix Exponentiation and Web Log Records", *Proc. 24th Int. ACM SIGIR Conference on Research and Development in Information Retrieval*, New Orleans, LA, Sep. 2001.

[20] A. J. Riel, *Object-Oriented Design Heuristics*, Addison-Wesley, Boston, MA, 1996.