Measuring Maintenance Effort in Object Oriented Software with Indirect Coupling

Nirmal Kumar Gupta
CSIS Department, BITS-Pilani
Pilani, Rajasthan
India

Mukesh Kumar Rohil
CSIS Department, BITS-Pilani
Pilani, Rajasthan
India

ABSTRACT
Measurement of maintenance effort in object oriented software engineering is one of the major challenges. Coupling among classes is one of the major factors determining the maintenance effort. Coupling is measured as strength of interconnection or interdependence between different parts of the Classes in object oriented software. It is widely accepted that there is strong relationship between high coupling and poor maintainability. Indirect coupling which is transitive in nature manifests between two seemingly unrelated parts of the system through hidden connections plays a major role in determining maintenance effort. This research proposes a set of metrics which determines maintenance effort for software with Indirect Coupling.

General Terms
Software Quality, Software Engineering, Object Oriented Tesing.

Keywords
Indirect Coupling, Software Maintenance Effort, Object Oriented Software, Software Quality.

1. INTRODUCTION
In Software Engineering, two design concepts (i.e. coupling and cohesion) are significant in developing good software [13]. Coupling is the degree to which one component is dependent on other software components. A component with high coupling value is highly interdependent on other software components and vice versa. If a component is highly interdependent then any change in the component requires significant changes in other components to which it is coupled [1]. Hence highly coupled components require high maintenance effort. It can be noted that a system cannot completely be devoid of coupling for the proper functioning of a system. There is some need of some connections among various sub components (classes) of a system. Hence maintaining loose coupling among components is desirable characteristics of good software design [2].

Object oriented software has various kinds of relationships among its components. Eder et al. [4] identify three different types of relationships. These relationships are interaction relationships between methods, component relationships between classes, and inheritance between classes, are then used to derive different dimensions of coupling which are classified according to different strengths. Hitz and Montazeri [5] approach coupling by deriving two different types of coupling: object level coupling and class level coupling which are determined by the state of an object and the state of an object's implementation, respectively. While our understanding of coupling is improving, most research has been applied only to direct coupling which is, coupling between classes that have some direct relationship [14]. However, there has been little investigation into indirect coupling, which is, coupling between classes that have no direct relationship [11]. The discussion in existing literature just implies that indirect coupling is little more than the transitive closure of direct coupling.

Various researchers have worked in this area and tried to address the high coupling relating it with the maintenance effort by minimizing effect of coupling by keeping the value of indirect coupling as low as possible [5]. This indirect form of coupling has a stronger impact over maintenance effort as compared to direct coupling. This increased maintenance effort is due to the effort required in tracing and modification of the software components. It leads to increase in effort required with the increase in length and number of connections between software components [6]. Therefore one must try to achieve the value of indirect coupling as low as possible.

Measurement of indirect couplings can be achieved in many ways [6]. The most basic measure of coupling involves simply counting the number of other classes to which a given class has a linkage. If a STUDENT studies in a UNIVERSITY and admitted to some COURSE, and then assuming STUDENT, UNIVERSITY and COURSE are three classes, STUDENT would have a coupling value of 2. By this measure, an understanding of which classes are most coupled within the system can be made.

In this paper, we will propose set of metrics which will relate maintenance effort with indirect coupling. We will first propose direct coupling and indirect coupling metric and then we will relate indirect coupling to maintenance effort. This gives us a clear idea, how indirect coupling affects maintenance effort. The rest of the paper is organized as follows. Sec. 2 discusses the related work done in measuring maintenance effort in object oriented software engineering through indirect coupling, followed by our proposed solution in Sec. 3. Section 4 talks about experimental results, and Sec 5 concludes our work.

2. RELATED WORK
Various researchers have put their efforts to define and measure various forms of couplings. According to Fenton and Pfleeger [7] "There are no standard measures of coupling". Many of the researches use some variation of Yourdon and Constantine's [8] original definition which defines as "a measure of the strength of interconnection" between modules. They suggests that coupling should be concretely defined in terms of the probability that coding, debugging, or modifying one module will necessary require consideration of changes of

another module. Such kind of definitions are not formal since they don't specify the meaning of "strength" or "interconnection". But such idea is an excellent heuristics for guiding the design of the software.

Berard [12] provides a new taxonomy to object-oriented coupling. He divides coupling into two basic categories: interface and internal coupling. Interface coupling exists when one object (client) makes use of another (server) object interface by calling its public method. In this case any change occurs to the interface of server object, should make correspondingly change in client object, but immune to any change occurs in the internals of classes of server object. Internal coupling occurs when an entity accesses an object's state, either from the "inside" or "outside". "Inside" Internal coupling occurs when that entity is a constituent of the accessed object, for example its method or a component object. "Outside" internal coupling occurs when that entity is a subclass or an unrelated object. He emphasizes that internal coupling is stronger and hence less desirable than interface coupling. Even more, outside internal coupling is always stronger than its counterpart inside internal coupling.

Yang et al. [6] defines direct coupling as: a given class C is directly coupled to another class D, if there is an explicit reference to class D within the source code of class C. Class C has explicit reference to class D, if class D is the declared type of any expression or sub-expression contained within the source code of class C [6]. Direct coupling has compilation dependency which makes the dependent classes to undergo recompilation whenever a change is made in the class on which they are depending for proper functioning of the system. Therefore any change in coupled class D will requires subsequent change in class C also, otherwise class C may not compile. On the other hand, Indirect coupling which is transitive in nature is defined as: if a class X is coupled to class Y, which is in turn coupled to class Z, then class X is transitively dependent on class Z and hence indirectly coupled. Thus a modification on Z may cause a cascading effect along the connections, i.e. ripple effect [4] if there are more number of classes involved between X and Z in general. However, Indirect coupling manifests through hidden connections between seemingly unrelated parts of software system. This means, that this type of coupling exists but not visible through direct connection. Therefore it is very important to investigate this aspect of coupling and its impact over the maintainability of a class which contributes to the overall quality of the software.

Eder et al. [4] gave another taxonomy of object-oriented coupling. He classifies coupling into three general forms: interaction, component and inheritance. Interaction coupling effectively refers to the following type of coupling: content, common, external, control, stamp and data coupling which are applied in the object oriented context, where the participants of coupling are methods and classes instead of modules. Component coupling concerns type usage relationship; class C is component coupled to C` if any of C's instance variable, local variables or method parameters of type C` are accessed by C. Component coupling represents compile time dependencies in the object oriented context. Finally, inheritance coupling refers to the inheritance relationship between a class and its direct or indirect subclass.

Chidamber and Kemerer [2] [9] were first to define the metrics called Coupling Between Objects (CBO) for object-oriented coupling. The coupling for a class is the number of classes to which it is coupled. A class is deemed to be coupled to another class if it accesses one or more of its variables or invokes at least one of its methods. The inheritance based coupling is ignored. They state that high value of CBO means that high coupling value which result in high maintenance effort and therefore should be avoided [15].

2.1 Limitations

Based on our literature study we identify following limitations within the established metrics.

1. There are various issues with the definition of CBO. One is that definition is not specific as to whether a couple is counted in terms of instances or the number of classes.
2. Indirect coupling or strength between any two classes is measured as multiplications of direct coupling values along the path. The value of indirect coupling or strength leads to decrease as the path leads to increase since indirect coupling is multiple of direct coupling. The value of indirect coupling is maximum when the path length is one and leads to decrease as the path length increase. Consequently, it signifies that maintenance effort required decreases as path length increases and will be maximum when the path length is one.
3. The established metrics indirectly depends only upon the longest path even if the multiple paths exist between any two classes. If there are multiple paths exist between any two classes, then the value of indirect coupling is measured as maximum of various independent or shared path. Indirect coupling does not depend upon the number of paths existing between two classes; rather it only depends upon the path with highest indirect coupling value. So established metrics do not take into account the number of paths or number of connections existing between two classes.

3. PROPOSED SOLUTION

In this section we will describe maintenance effort in object oriented software engineering through indirect coupling. Indirect coupling can be related with an analogy. Like in real life, you ask for your friend for particular task, which in turn asks to his friend so that the task which you have assigned to your friend could be completed. It means, there is direct relationship between you and your friend and also there is direct relationship between your friend and his friend. So, there is no direct relationship between you and your friend's friend, but there is transitive relationship or indirect relationship between you and your friend's friend. In sense, there is effort in conveying the task to your friend's friend. Although, this relation seems to be hidden, but exists in real life. So, effort required to complete the task which you have assigned to your friend will be more than if it would had been done by your friend.

3.1 Direct Coupling

The coupling metric that takes account of the degree of coupling, functional complexity and transitive (i.e. indirect) coupling between classes, an object-oriented software system can be regarded as a directed graph [3]. The classes comprising the system are the vertices in the graph. Suppose such a system comprises a set of classes $C \cong \{C_1, C_2 \ldots . C_m\}$. Let M_j be the set of methods of the class C_j, and V_j be the set of instance variables of class C_j. $MV_{j,i}$ is the set of methods and instance variables in class C_i invoked by class C_j for $j \neq i$ ($MV_{j,j}$ is defined to be null). Then the edge from C_j to C_i exists if and only if $MV_{j,i}$ is not null. Thus an edge of the graph reflects the direct coupling of one class to another. The graph is directed since $MV_{j,i}$ is not necessarily equal to $MV_{i,j}$.

MV_j, the set of all methods and instance variables in other classes that are invoked by class C_j, can be defined as [3]:

$$\mathrm{MV_j} = \bigcup_{1 \le i \le m} \mathrm{MV_{j,i}} \qquad (1)$$

The extent of direct coupling from class C_i to class C_j depends upon the number of methods and variables in the set $MV_{i,j}$. The class is said to be coupled strongly if this value is large.

Using the above notations we can define direct coupling from class C_i to class C_j as [3]:

$$C_D^{i,j} = \frac{|MV_{i,j}|}{|MV_i| + |M_i| + |V_i|} \qquad (2)$$

In the above equation denominator represents the total number of methods and variables used by class C_i, which accounts for the total functionality of class C_i. This guarantees that the direct coupling from class C_i to class C_j, $C_D^{i,j}$ is independent of class size. As per the definition in equation (2) the value of $C_D^{i,j}$ will always be in the range from zero to one.

3.2 Indirect Coupling

Suppose that the two direct coupling values $C_D^{i,j}$ and $C_D^{j,k}$ for classes C_i, C_j and C_k, but the value of direct coupling $C_D^{i,k}$ is zero. Even though there is a dependency between classes C_i and C_k because C_i is depending upon C_j which in turn depends upon class C_k. Because of this indirect dependency any modification done in class C_k may affect class C_i. Therefore at the time of maintenance activities such indirect relations must be considered and the maintenance effort will depend upon the coupling path between C_k and C_i. This maintenance effort depends upon the fact that how strongly the individual classes are coupled together. Therefore we can define this effort as:

$$E_I^{i,k} = E\left(C_D^{i,j} + C_D^{j,k}\right) \qquad (3)$$

A coupling between two classes exists if there is a path from one to the other made up edges whose direct coupling values C_D are all non-zero. The maintenance effort required depends upon the sum of all those C_D values. Thus we define effort required because of this indirect coupling between classes C_i and C_j due to a specific path *p*, as:

$$E_I^{i,k}(p) = E\left(\sum_{e_{s,t} \in p} C_D^{s,t}\right) = E\left(\sum_{e_{s,t} \in p} \frac{|MV_{s,t}|}{|MV_s| + |M_s| + |V_s|}\right) \qquad (4)$$

Here $e_{s,t}$ denotes the edge between vertices *s* and *t*.

We are assuming that coupling between any two classes will always be less than 1, so indirect dependency due to longer paths will lead to increase. Longer will be the path, higher will be the indirect coupling and vice versa. Here we measure Maintenance effort in terms of value of indirect coupling which is directly proportional to indirect coupling.

Effort α Indirect Coupling

Higher the value of indirect coupling, greater maintenance effort required in tracing and modification and vice versa.

3.2.1 Relationship between Indirect Coupling and Maintenance Effort

Effort α Σp∈ Chains length(p)

Effort [10] is associated with the sum of direct coupling measured along each path in the set of paths or chains exist among different software components (classes) [2]. This theory states that Effort is directly proportional to length of the path. This theory can be related by extending our previous analogy in which we has completed our task by delegating to your friend's friend. If we extend the same analogy to one more level, then the effort required will increase rather if it had been done by your friend. Similarly, if our path will increased by one, then indirect coupling will increase which is sum of direct coupling along the path, so maintenance effort required to trace or modification or change will increase cost.

Effort is associated with the sum of lengths (measured in terms of edges) of all chains in the set Edges. This theory states that greater is the length of the chain, more effort will require to trace.

Effort α Number of Chains

Effort is associated with the number of chains or set of paths. This theory states that Effort is directly proportional to number of chains or path exists between different software components (classes). This theory can be related by extending our previous analogy in which we assume that there is only one path to get the task completed by your friend's friend. Now, we assume that there is another task which may or may not through overlapping route, but the destination is same. So, there is multiple path exists between source and destination. Even, the length of path may or may not be same. Since, multiple path or relationship exists between source and destination, so effort required in getting the work done will be more. Similarly if there is multiple paths exist between different software component, which may or may not have overlapping edge in common and even path length may vary, then maintenance effort required in tracing the path and modification will costs more.

Thus, effort required in this situation can be determined using the following algorithm. This algorithm considers that if there are multiple paths between two classes and partially they are overlapping then redundant effort for such common edges must be removed.

Assume for the two classes C_i and C_j various coupling paths p_s (s > 0) exist and partially they may be overlapping. Each path p_s consists of a certain number of edges e_{ks}, $(1 \le k \le n_s)$. Each edge is formed between two classes if they are directly coupled. Here ns is the total number of edges along path ps. If the maintenance effort required along edge e_{ks} is denoted by $E(e_{ks})$ then the algorithm can be written as:

```
Initialize E_o=0
for each path p_s
    initialize for all r_k = false
    for each edge e_ks in path p_s
        if r_k == false
            E_o(new) = E_o(old) + E(e_ks)
            r_k = true
        endif
    endfor
endfor
```

Fig 1: Algorithm for computing maintenance effort when multiple, possibly overlapping coupling paths exist between two classes.

Thus increasing the number of paths or connections between different software components (classes) will increase indirect coupling productively which resulting in increase in the maintenance effort required in modification or extending the functionality of the class and tracing the path.

4. EXPERIMENT

We consider the case study with EasyMock v3.1 software for validation of our proposed metrics. EasyMock is open source library that provides an easy way to use Mock Objects for given interfaces. It helps to create mock objects which can be easily used in conjunction with JUnit. We have used a set of 37 classes from the source code of EasyMock. We identified coupling relations of different lengths between these classes. A coupling length of more than one involves more than two classes. The number of classes identified for different coupling lengths are summarized in Table 1. We have considered coupling lengths from 2 to 5 and identified the number of such paths existing in our class cluster.

Table 1. Number of paths and total number of classes involved for different path lengths.

Path length (n)	Total Number of classes involved	Number of paths (N_n)
2	36	65
3	27	46
4	18	31
5	11	19

Table 2 summarizes the overall coupling of all individual paths having a path length of 2. The overall coupling along a path must be computed in such a manner that it must represent the overall effort required during maintenance activities following that path. This maintenance effort increases in an incremental manner along with the path length. Here p denotes the path number; C_{e1} and C_{e2} are coupling values of individual edges along the path and is computed using equation (2). C_o represents the overall coupling for the corresponding path.

Table 2. Summarizing coupling values for each path for a path length of 2.

p	C_{e1}	C_{e2}	C_o	p	C_{e1}	C_{e2}	C_o
1	0.39	0.37	0.76	34	0.14	0.28	0.42
2	0.37	0.38	0.75	35	0.26	0.10	0.36
3	0.44	0.32	0.76	36	0.10	0.06	0.16
4	0.04	0.62	0.66	37	0.39	0.23	0.62
5	0.58	0.47	1.05	38	0.55	0.66	1.21
6	0.04	0.29	0.33	39	0.51	0.37	0.88
7	0.49	0.13	0.62	40	0.38	0.69	1.07
8	0.29	0.15	0.44	41	0.43	0.44	0.87
9	0.55	0.43	0.98	42	0.61	0.29	0.9
10	0.38	0.47	0.85	43	0.18	0.34	0.52
11	0.04	0.20	0.24	44	0.44	0.31	0.75
12	0.26	0.42	0.68	45	0.20	0.06	0.26
13	0.18	0.21	0.39	46	0.37	0.38	0.75
14	0.44	0.14	0.58	47	0.48	0.73	1.21
15	0.49	0.31	0.80	48	0.43	0.09	0.52
16	0.13	0.42	0.55	49	0.34	0.01	0.35
17	0.07	0.38	0.45	50	0.50	0.64	1.14
18	0.40	0.34	0.74	51	0.19	0.66	0.85
19	0.42	0.34	0.76	52	0.53	0.16	0.69
20	0.29	0.07	0.36	53	0.38	0.29	0.67
21	0.16	0.10	0.26	54	0.19	0.58	0.77
22	0.56	0.15	0.71	55	0.33	0.48	0.81
23	0.20	0.14	0.34	56	0.61	0.67	1.28
24	0.23	0.34	0.57	57	0.33	0.35	0.68
25	0.75	0.45	1.20	58	0.60	0.21	0.81
26	0.71	0.03	0.74	59	0.35	0.63	0.98
27	0.20	0.16	0.36	60	0.39	0.60	0.99
28	0.34	0.27	0.61	61	0.18	0.24	0.42
29	0.46	0.49	0.95	62	0.59	0.27	0.86
30	0.56	0.53	1.09	63	0.20	0.31	0.51
31	0.36	0.25	0.61	64	0.33	0.54	0.87
32	0.55	0.45	1.00	65	0.50	0.65	1.15
33	0.48	0.17	0.65				

$$\text{Mean of } C_o = \frac{\sum C_o}{N_2} = \frac{46.17}{65} = 0.71$$

Similarly we have Table 3 which summarizes the overall coupling of all paths having a path length of 3. We also compute mean of coupling values computed of all paths for different path lengths.

Table 3. Summarizing coupling values for each path for a path length of 3.

p	C_{e1}	C_{e2}	C_{e3}	C_o	p	C_{e1}	C_{e2}	C_{e3}	C_o
1	0.31	0.46	0.40	1.17	24	0.26	0.63	0.81	1.70
2	0.49	0.56	0.59	1.64	25	0.70	0.35	0.03	1.08
3	0.57	0.15	0.14	0.86	26	0.27	0.11	0.69	1.07
4	0.95	0.30	0.34	1.59	27	0.11	0.28	0.52	0.91
5	0.13	0.13	0.67	0.93	28	0.47	0.20	0.37	1.04
6	0.40	0.32	0.30	1.02	29	0.78	0.09	0.76	1.63
7	0.42	0.39	0.37	1.18	30	0.68	0.01	0.37	1.06
8	0.12	0.64	0.47	1.23	31	0.47	0.11	0.25	0.83
9	0.53	0.57	0.59	1.69	32	0.18	0.76	0.17	1.11
10	0.78	0.71	0.48	1.97	33	0.57	0.96	0.25	1.78
11	0.56	0.58	0.83	1.97	34	0.71	0.50	0.55	1.76
12	0.21	0.76	0.29	1.26	35	0.07	0.40	0.54	1.01
13	0.55	0.80	0.40	1.75	36	0.64	0.75	0.10	1.49
14	0.60	0.32	0.15	1.07	37	0.15	0.59	0.42	1.16
15	0.64	0.47	0.30	1.41	38	0.51	0.16	0.44	1.11
16	0.61	0.65	0.70	1.96	39	0.67	0.28	0.52	1.47
17	0.63	0.51	0.81	1.95	40	0.27	0.23	0.61	1.11
18	0.50	0.66	0.46	1.62	41	0.25	0.19	0.76	1.20
19	0.20	0.54	0.60	1.34	42	0.31	0.30	0.70	1.31
20	0.21	0.44	0.18	0.83	43	0.20	0.65	0.33	1.18
21	0.65	0.49	0.16	1.30	44	0.76	0.96	0.82	2.54
22	0.03	0.75	0.03	0.81	45	0.04	0.37	0.63	1.04
23	0.34	0.66	0.64	1.64	46	0.60	0.68	0.68	1.96

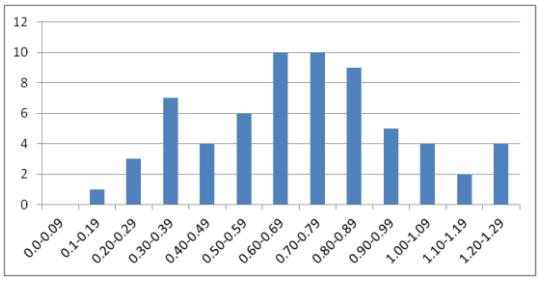
$$\text{Mean of } C_o = \frac{\sum C_o}{N_3} = \frac{62.74}{46} = 1.36$$

In Table 4 we summarize the mean values of completed overall coupling values for different path lengths.

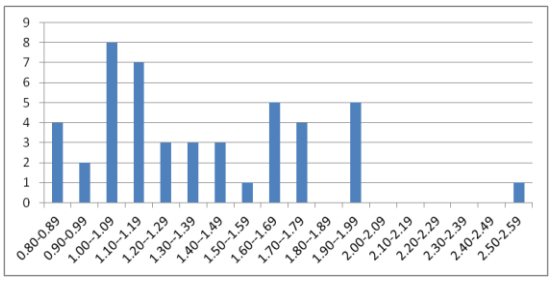
Table 4. Mean values of overall coupling values for different path lengths.

Path length (n)	2	3	4	5
Mean of C_o	0.71	1.36	1.95	2.72

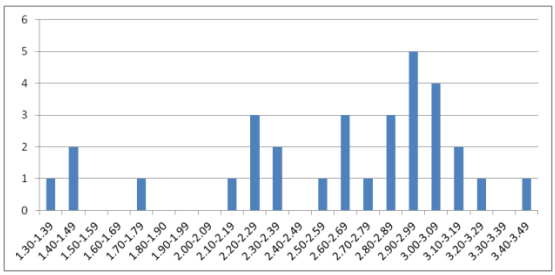
In Fig.2 we draw the frequency of paths in paths in different coupling slots for path length of 2, 3, 4 and 5. In Fig.2(a) we observe that there are more number of paths having a coupling value in the middle range. If there are more number of paths in the higher coupling range it would mean more maintenance effort will be needed.



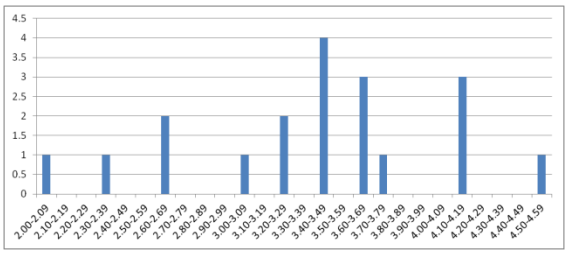
(a)



(b)



(c)



(d)

Fig 2: Frequency of paths in different coupling slots for (a) path length of 2 (b) path length of 3 (c) path length of 4 (d) path length of 5

Since here it is in the middle range comparatively less maintenance effort will be needed for classes which are coupled with coupling path length of 2. In Fig 2(b) many paths are in the lower coupling range that means overall maintenance effort will be less for path length 3 in taken case study. Similarly in Fig 2(c) and Fig 2(d) we observe that coupling range for C_o is larger and most of the paths are having larger values and fall at the end in these figures. Therefore, maintenance effort in this case is larger for classes coupled with these path lengths.

5. CONCLUSION

In this research we have identified some limitations of established coupling measurement metrics which measure the indirect coupling between any pair of classes in a class cluster. We have provided our metrics which is in relation with the maintenance effort required for these classes. In our experiment it is shown that as the path length increases between two classes the value of indirect coupling also increases as it is expected since the maintenance effort must increase. This increase in Indirect coupling value depends upon how strongly the classes are coupled together. Strongly coupled are expected to require more effort at the time of maintenance.

6. REFERENCES

[1] Briand, L., Daly W. and Wust J., 1999, A Unified Framework for Coupling Measurement in Object-Oriented Systems. IEEE Transactions on software Engineering, Vol. 25, 91-121.

[2] Chidamber S.R. and Kemerer C.K., 1991, Towards a Metrics Suite for Object Oriented Design, Proceedings of 6th ACM Conference on Object Oriented Programming, Systems, Languages and Applications (OOPSLA'91), (Phoenix, Arizona, 1991), 197-211.

[3] Dallal J. and Briand L., 2010, An object-oriented high-level design-based class cohesion metric, International Software Technology, 52 (12), 1346-1361.

[4] Eder, J., Kappel G. and Schrefl M., 1994, Coupling and cohesion in object-oriented system, Technical report, Univ. of Klagenfurt.

[5] Hitz H. and Montazeri B., 1995, Measuring Coupling and Cohesion In Object-Oriented Systems, Proc. Int'l Symp. Applied Corporate Computing (ISACC '95), Monterrey, Mexico, Oct. 25-27.

[6] Yang H. and Tempero E., 2007, Measuring the Strength of Indirect Coupling, In Proceedings of the 2007 Australian Software Engineering Conference (ASWEC '07). IEEE Computer Society, Washington, DC, USA, 319-328.

[7] Fenton N.E. and Pfleeger S.L., 1997, Software Metrics - A Rigorous & Practical Approach, *ITP London*, (1997).

[8] Yourdon E. and Constantine L., 1979, Structured Design: Fundamentals of a Discipline of Computer Program and System Design. Prentice-Hall.

[9] Chidamber S.R. and Kemerer C.K., 1994, A Metrics Suite for Object Oriented Design. IEEE Transactions on Software Engineering, Vol. 20 (June 1994), 476-493.

[10] Slaughter S., Harter D. and Krishnan M., 1998, Evaluating the Cost of Software Quality, Communications of the ACM, 41 (8), 67-73.

[11] Yang H. and Tempero E., 2007, Indirect Coupling as a Criteria for Modularity. In Proceedings of the First International Workshop on Assessment of Contemporary Modularization Techniques (ACoM '07). IEEE Computer Society, Washington, DC, USA, 10-11.

[12] Berard E., 1993, Essays on Object-Oriented Software Engineering, volume 1, chapter 7. Prentice Hall, Englewood Cliffs, New Jersy.

[13] Gui G. and Scott P.D., 2006, Coupling and cohesion measures for evaluation of component reusability. In Proceedings of the 2006 international workshop on Mining software repositories (MSR '06). ACM, New York, NY, USA, 18-21.

[14] Li, W., and Henry, S. 1993. Object-oriented metrics that predict maintainability. J. Systems and Software 23(2), 111–122.

[15] Card, D. N., Church, V. E., and Agresti, W. W. 1986. An empirical study of software design practices. IEEE

[16] Transactions on Software Engineering 12(2): 264–271.