Middle-East Journal of Scientific Research 23 (5): 801-807, 2015
ISSN 1990-9233
© IDOSI Publications, 2015
DOI: 10.5829/idosi.mejsr.2015.23.05.22163

Run Time Coupling Measurement in a Distributed Object Oriented Systems

S. Babu

IFET College of Engineering, Gangarapalayam, Tamilnadu, India

Abstract: Software metrics play a key role in the planning and in the control of software development projects. The Estimation of Coupling helps in the development of quality Software Products as well as in the maintenance. They reason out the structural complexity of software and to envisage the quality of the Software Product. Quality attributes such as Fault-proneness, ripple effect of changes and changeability are well predicted by coupling measures. Coupling or Dependency states the degree to which each individual program module relies on each one of the other modules. Coupling measures portray the static dependency among the classes in an object-oriented system. "Dynamic" couplings are not taken into an account due to polymorphism and may notably take too lightly the intricacy of software and misjudge the need for code inspection, testing and debugging. Most probably this results in hapless predictive exactness of the quality models that utilize static coupling measurement. In the proposed method, there are three steps such as Instrumentation process, Post processing and Coupling measurement.

Key words: Software Metrics · Object Oriented Systems · Static Coupling · Dynamic Coupling

INTRODUCTION

The prominent goal of software engineering is building software systems that function as expected and are of high quality with smallest amount. The idea of excellence in a software system hinges on a variety of facets, such as how many defects it contains, easy it is for developers to inadvertently inject faults into the system and the complexity concerned in modifying and understanding all or parts of the system. Ensuring excellence is vital as for one thing it naturally ties with the cost of developing and maintaining the system. Over the past few decades there has been much research on understanding what is meant by excellence and extra prominently, how suitable quality can be achieved and maintained [1]. Measurement provides a means to objectively assess the properties of the product and evaluate whether it meets its desired qualities.

There is an increasing awareness on the importance of software measurement within the software manufacturing society, as well as the necessity of respecting the scientific basis of dimension. But there is small proof for the latter as there is a tendency for researchers and practitioners to apply software metrics without a full awareness of what they mean. Coupling is defined as the measure of interdependency or interconnectivity between various modules of a software system, which is one important property for which many metrics have been clear. While it is extensively approved that there is a relationship between high coupling and poor maintainability, current empirical evidence towards this is insufficient to promote a full understanding of this relationship. Software metric is a measure of some property of a piece of software or its specifications. Coupling is one of the software metric for object oriented software [2].

Related Work: Software metrics plays a key role in the planning and in the control of software development projects. The Estimation of Coupling helps in the development of quality Software Products as well as in the maintenance. They reason out the structural complexity of software and to envisage the quality of the Software Product. Quality attributes such as Fault-proneness, ripple effect of changes and changeability are well predicted by coupling measures. Coupling or Dependency states the degree to which each individual program module relies on each one of the other modules. The Coupling measures portray the static dependency among the classes in an object-oriented system [3].

Corresponding Author: S. Babu, IFET College of Engineering, Gangarapalayam, Tamilnadu, India.

Dynamic couplings are not taken into an account due to polymorphism and may notably take too lightly the intricacy of software and misjudge the need for code inspection, testing and debugging. Most probably this results in hapless predictive exactness of the quality models that utilize static coupling measurement.

Coupling: Coupling analysis is one of the diverse methods used in software system for modeling and measuring the relationships between components. Two components having any type of connection or relationship between them are coupled by coupling analysis. Coupling is described as a "measure of the strength of interconnection between modules". Now the term potency and interconnection lead to a multitude of interpretations, evident in the several coupling metrics introduced in the literature. Software metric is a measure of some property of a piece of software or its specifications. Coupling is one of the software metric for object oriented software [4].

Dynamic Coupling: Coupling indicates the strength of association established by a connection from one module to another. Traditional coupling measures take into account only static couplings. They do not version for dynamic couplings due to polymorphism and may significantly underestimate the complexity of software and misjudge the need for code inspection, testing and debugging. To address this problem defined a family of dynamic coupling measures that account for polymorphism [5].

Coupling measures have important applications in software development and maintenance. They are used to help developers, testers and maintainer's cause regarding software complexity and software quality attributes. Couplings have been used to assist maintainers in tasks such as impact analysis, assessing the fault-proneness of classes, fault prediction, ripple effects and changeability among others [2]. Coupling or dependency is the degree to which each program module relies on each one of the other modules [6]. Coupling measures set apart the static and dynamic usage dependencies among the classes in an object-oriented system.

Dynamic coupling is more precise than static coupling for systems with unused code, which is uninteresting in most situations and can seriously bias analysis [7]. They conducted a number of studies on the quantification of many runtime class-level coupling metrics such as Dynamic Coupling between Objects (DCBO) for object-oriented (Java) programs. They also statistically analyzed the differences in the underlying dimensions of coupling captured by static versus dynamic coupling metrics using Chidamber and Kemerer (CK) suite metrics like Coupling between Objects (CBO) and Lack of Cohesion in Methods (LCOM). The results indicated that the run-time metrics did capture different properties than the static metrics. Dynamic coupling metric suite for service-oriented systems that evaluates service's quality according to its ability of coupling [8].

Coupling Measurement in Object-Oriented Systems: Object oriented technology is gaining significant importance in the field of software development. To evaluate and maintain the quality of object-oriented software where there is a need to assess and analyses its design and implementation using appropriate measurement metrics. A quality metric should relate to external quality attributes of design which include maintainability, reusability, error proneness and understandability. Coupling has shown a direct impact on software quality. In general, one of the goals of software designers is to keep the coupling in object-oriented system as low as possible [10].

Distributed Object Oriented Systems (DOOS): The distributed system is one in which components located at networked computers and devices communicate and coordinate their actions by message passing for sharing resources and for distributing computational workload [11]. In a distributed object oriented system, service providers are distributed across a network of systems and are called servers. Clients, who issue requests for those services, are also distributed across a network. A single program can be both a client (issue request) and a server (service request). Clients can issue requests to local and remote servers. The large numbers of Distributed Object Oriented (DOO) systems have been developed for solving complex problems in various scientific fields [10]. In such a case, the DOO software may need to be restructured. The challenge resides in the complexity of interactions between objects [12].

Dynamic Coupling Measurement: Works available in the literature for software metrics have mainly concentrated on centralized systems. Only very few of them have focused on distributed systems and more specifically on service-oriented systems. Conventional non distributed systems differ from systems with distributed components in several ways including communication type, latency, concurrency, fractional, versus entire failure and

referencing parameters passing strategies. Normally, distributed systems with service oriented components are more complex because they accomplish efficiency and other quality characteristics in a more heterogeneous networking and implementation environment.

Traditional coupling measures take into account only "static" couplings. They do not relation for "dynamic" couplings [12] due to polymorphism and may significantly underestimate the complexity of software and misjudge the need for code examination, testing and debugging. This is expected to result in poor predictive accuracy of the quality models in distributed Object Oriented systems that utilize static coupling. In order to overcome these issues, we propose a model in Distributed Object Oriented Software to measure the coupling dynamically via three stages.

Instrumentation Process: First, the instrumentation process is performed. Within this procedure, to trace technique calls, for the duration of this process, three trace files, prf. clp. and svp. are created.

Post Process: In the second step, the information present inside these files, are merged.

Coupling Measurement: At the end of this step, the files are merged. Finally, the coupling metrics are measured dynamically [13].

Measuring Dynamic Coupling in Doo Systems: In this research work, we propose a model in Distributed object oriented Software for coupling measurements dynamically. In this model, there are three steps such as Instrumentation process, Post processing and measurement of coupling. Initially the instrumentation process is done. In this process the instrumented Java Virtual Machine (JVM) that has been modified to trace the method calls. In this process, 3 trace files are formed namely prf. clp. svp. In the second step, the data's in these file are combined. At the end of this step, the combined details trace of each JVM contains pointers to the merged trace files of the other JVMs such that the path of every remote call from the client to the server can be exclusively identified. Finally, the coupling metrics are calculated dynamically. The brief explanation of the proposed method is described as follows [14].

Introspection: In object-oriented languages, introspection is the idea that program code can get the details on itself. In OO languages, introspection permits programs to get and use information on classes and objects at Execution-time. By introspection, one can request an object used for its class, ask a class for its methods and constructors find out the details of those methods and constructors and tell those methods to perform. Introspection is or else known as instrumentation process. The java class files are executed by instrumented JVM to generate the execution events. Introspection allows the main application to examine each of its plug-ins for the methods it supports and then call them whenever appropriate. For the introspection to happen, we first compiled the source code and then used reflection to retrieve data members and methods. Later vectors are used to store the retrieved information.

Trace Events: The trace generation module of the JVM is modified to record every invocation of a method using time stamps that show the start and end times of the method with microsecond resolution. As a Java program is executed by the instrumented java virtual machine, 3 trace files are generated, i.e., prf. clp. and svp. files [15].

Profile File (.Prf File): The prf. file contains detailed trace information that records call and return time stamps for every process executed. Invocations of the similar method executed under different threads are distinguished from one another by their unique thread identifiers. The other two files record the client or server communication, if any, that occurs on the java virtual machine as the program is being executed.

Clipboard File (.Clp File): The clp. file contains information about all of the outgoing Remote Method Invocation (RMI) calls from the running java virtual machine, i.e., identifying information for remote methods invoked by this JVM.

Server Profile File (.Svp File): The svp. file records information about all incoming RMI calls, i.e., all of the methods remotely invoked on this java virtual machine by other java virtual machines. The clp. file is referred to as the client profile of remote methods for which the JVM acts as a client and the svp. file is referred to as the server profile of remote methods for which the JVM acts as a server [16].

Merging: The major part of the merge procedure is to link client calls recorded in the client profile of a client JVM to the appropriate entry in the server profile of the remote server JVM where the method was actually executed.

Estimating the Dynamic Behavior: Because of the increasing complexity of the systems, it is often hard for software developers to clearly understand the behavior of the application. When the system is deployed and running and the performance goals are not met, it is difficult, if not impossible to determine, where problems are. Tracing a program is used to understand how the program executes and reveals the dynamic details of the design. After Events are added, the code along with the trace events is compiled and executed. During runtime, the actual function calls (*Ma*) are extracted based on the added trace events *(Mt)* given by $Ma = M \cap Mt$.

where *M* is the Number of classes invoking the methods of class C.

Dynamic metrics can be used as improving agents for static metrics by extracting the real time behavior of a software at runtime and taking the appropriate steps (such as eliminating a class that is never used at runtime).

RESULTS AND DISCUSSIONS

The coupling metrics for Distributed Object Oriented Systems, which is proposed in this research work, is implemented in the working platform of JAVA (version JDK 1.6). The results obtained from the proposed method are described as follows.

The sample empirical evidence in Table 6.1 resulted in reduction of 3 packages in coupling dependencies. Figure 6.1 and Figure 6.2 show the sample test run for static and dynamic coupling measurements respectively. These figures show, the number of packages used in both static and dynamic coupling. In this, the packages are used by both client and servers.

We tested the sample code (refer Table 5.2) with 25 continuous test runs as well as discrete test runs. Discrete test runs tend to produce 36% average coupling reduction but which might be due to arbitrary behavior of coupling reduction, whereas continuous test runs for 25 iterative executions resulted in 41.6% average reduction in coupling. We have also conducted the experiments for more iteration and the results support the claim that coupling is reduced, be it discrete or continuous.

The experiments were conducted on set of trivial java programs which involved a looping structure to decide on the number of iterative executions. Probably this may be the reason for deciding on the reduction in coupling around mid-value of x-axis which controls the number of continuous test runs. However, as the volume of the executions grows; the sudden change in the behavior of coupling reduction is minimized to a greater extent.

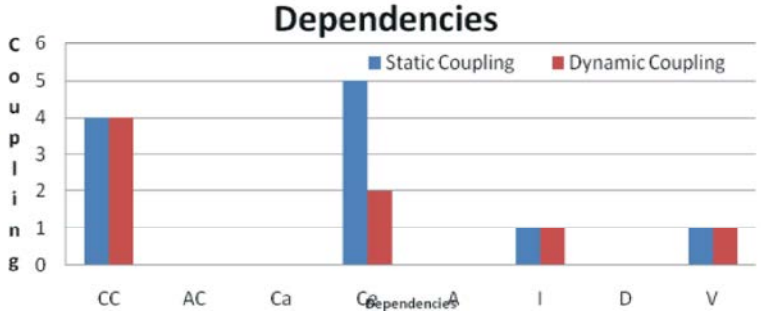

Fig. 1: Dependency Graph for 1 sample run

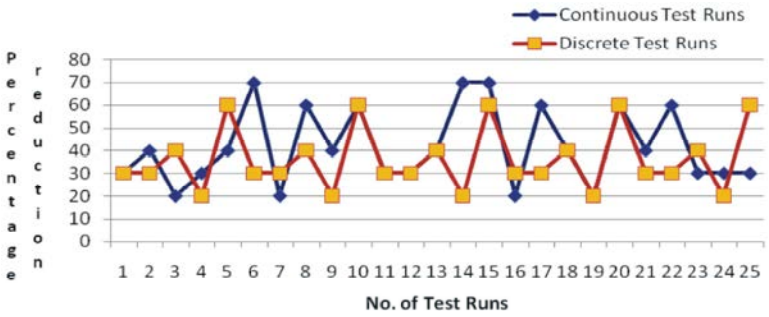

Fig. 2: Reduction in Coupling for 25 runs

Table 1: Comparison of Dependencies for 1 sample run

Dependencies	Static Coupling	Dynamic Coupling
CC	4	4
AC	0	0
Ca	0	0
Ce	5	2
A	0	0
I	1	1
D	0	0
V	1	1

Table 2: Comparison of Dependencies for 25 runs

Test run	Continuous Test Runs	Discrete Test Runs	Test run	Continuous Test Runs	Discrete Test Runs
1	3	3	14	7	2
2	4	3	15	7	6
3	2	4	16	2	3
4	3	2	17	6	3
5	4	6	18	4	4
6	7	3	19	2	2
7	2	3	20	6	6
8	6	4	21	4	3
9	4	2	22	6	3
10	6	6	23	3	4
11	3	3	24	3	2
12	3	3	25	3	6
13	4	4			

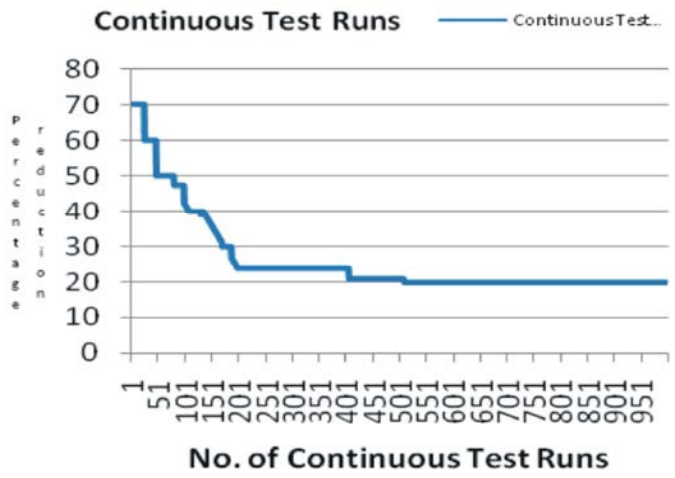

Fig. 3: Reduction in Coupling for 10^3 continuous runs

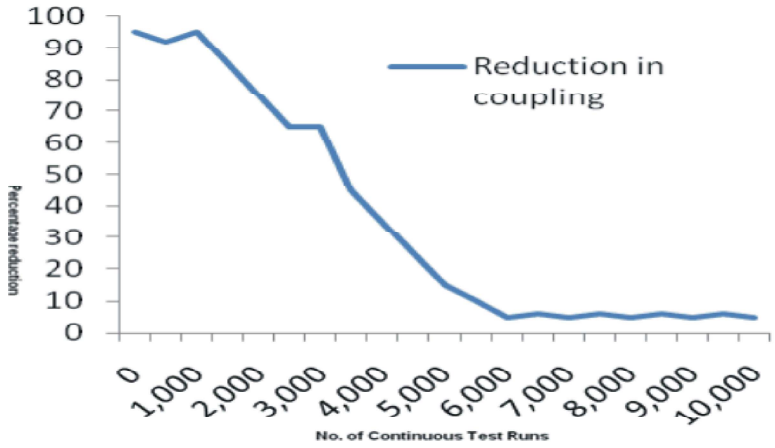

Fig. 4: Reduction in Coupling for 10^4 continuous runs

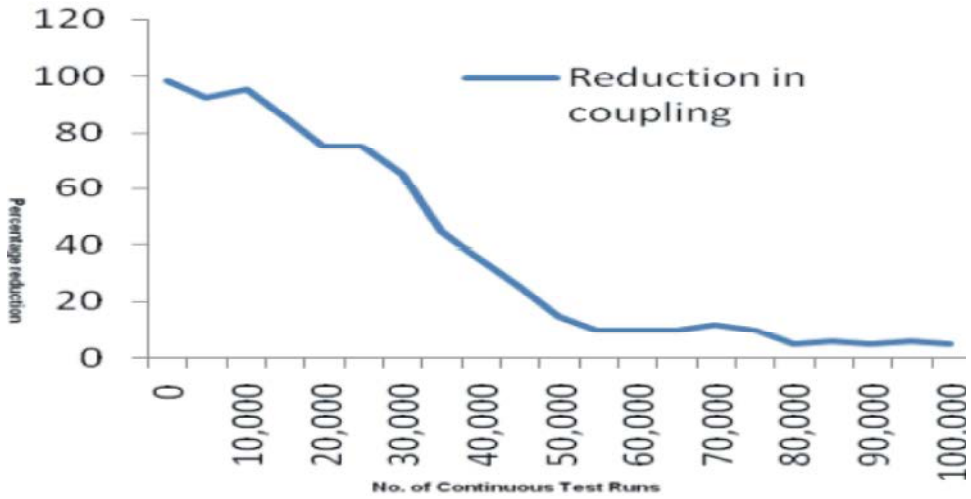

Fig. 5: Reduction in Coupling for 10^5 continuous runs

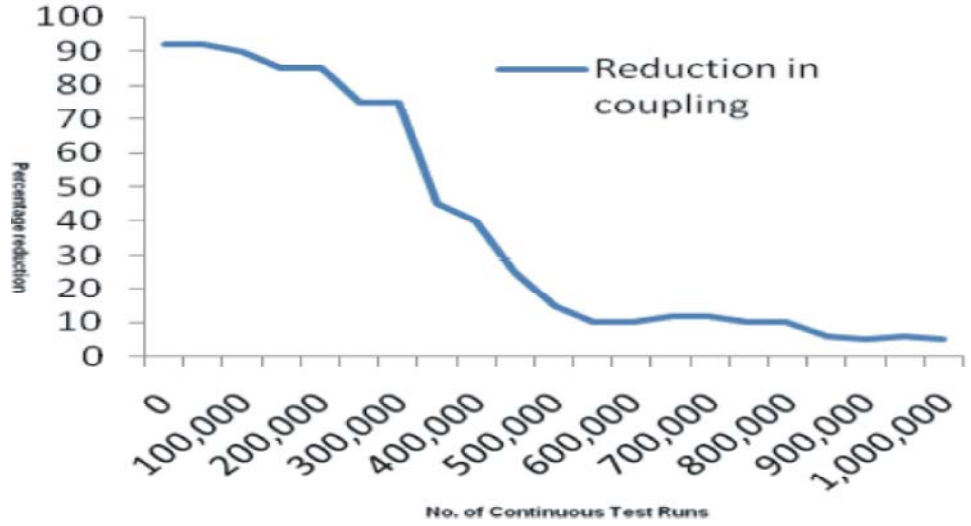

Fig. 6: Reduction in Coupling for 10^6 continuous runs

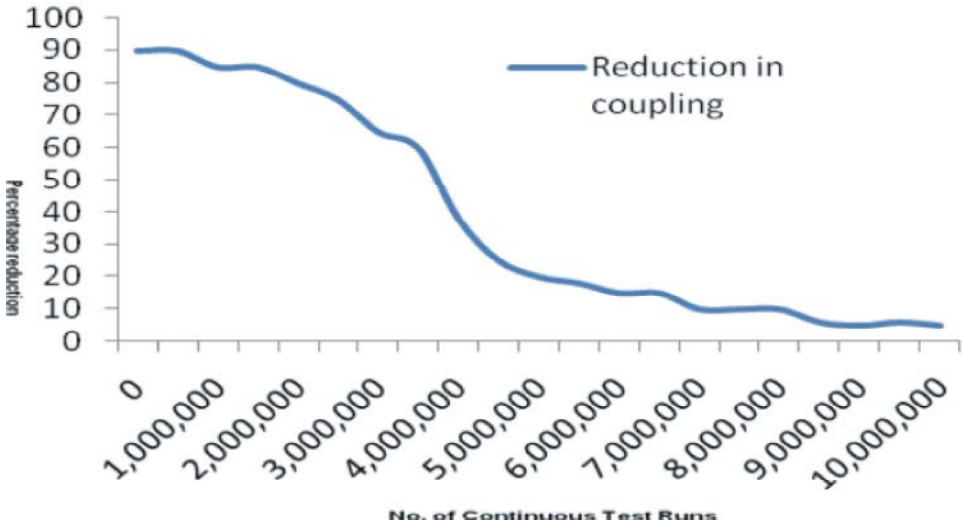

Fig. 7: Reduction in Coupling for 10^7 continuous runs

CONCLUSION

In this research work, it has been proposed that a new approach to the computation of dynamic coupling measures in DOO systems by introspection and adding trace events into methods. First, we provide formal, operational definitions of coupling measures and analysis. We propose dynamic coupling measures for distributed object-oriented systems i.e., coupling measurement on both clients and server dynamically. We described the categorization of dynamic coupling measures. The motivation for those measures is to complement existing measures that are based on static analysis by actually measuring coupling at runtime in the hope of obtaining better decision and prediction models because we account precisely for inheritance, polymorphism and dynamic binding. Admittedly, many other applications of dynamic coupling measures can be envisaged. However, investigating change proneness was used here to gather initial but tangible evidence of the practical interest of such measures.

Finally we propose our dynamic coupling measurement techniques which involve Introspection Procedure, adding together trace events into methods of all classes and Predicting Dynamic Behavior while running the source code. The source code is filtered to arrive the Actual Runtime used Source Code which is then given for any standard coupling technique to get the Dynamic Coupling.

REFERENCES

1. Mitchell Aine and James F. Power, 2006. A study of the influence of coverage on the relationship between static and dynamic coupling metrics, Science of Computer Programming, 59(1-2): 4-25.
2. Mitchell Aine and James F. Power, 2003. Toward a definition of run-time object-oriented metrics, 2003. In Proceedings of 7th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE'2003).
3. Arisholm, E., L. Briand and A. Foyen, 2004. Dynamic coupling measurement for object- oriented software, IEEE Trans. Software Engineering, 30(8): 491-506.
4. Arisholm Erik, 2002. Dynamic Coupling Measures for Object-Oriented Software, In proceedings of 8th IEEE Symposium on Software Metrics (METRICS'02), pp: 33-42.
5. Gurunadha Rao Goda and Avula Damodaram, 2011. Measurement of Dynamic Coupling in a Object Oriented System Based on Trace Events, American J. Scientific Research ISSN 1450-223, 10(7): 43-55.
6. Briand, L.C., J. Daly, V. Porter and J. Wust, 2000. A Comprehensive Empirical Validation of Product Measures for Object-Oriented Systems, Fraunhofer Inst. for Experimental Software Engineering, Germany, Technical Report ISERN-98-07.
7. Choi Misook and Jong Suk Lee, 2007. A Dynamic Coupling for Reusable and Efficient Software System, 5th ACIS International Conference on Software Engineering Research, Management & Applications, pp: 720-726.
8. Singh Paramvir and Hardeep Singh, 2010. Class-level Dynamic Coupling Metrics for Static and Dynamic Analysis of Object-Oriented Systems, International Journal of Information and Telecommunication Technology, 1(1): 16-28.
9. Chin, R.S. and S.T. Chanson, 1991. Distributed Object-based programming systems, ACM Computing Surveys, 23(1): 91-124.
10. Hamad, Safwat H. Reda A. Ammar, Mohammed E. Khalifa and Ayman El-Dessouky, 2008. A Multi step Approach for Restructuring and Mapping Distributed Object-Oriented Software onto a Multiprocessor System", In proc. of the INFOS2008, March 27-29, 2008 Cairo-Egypt, pp: 19-23.
11. Yacoub, Sherif M. Hany H. Ammar and Tom Robinson, 2002. Dynamic Metrics for Object Oriented Designs, Software Metrics Symposium, Proceedings, 6: 50-61.
12. Babu, S. and R.M.S. Parvathi, 2012. Development of Dynamic Coupling Measurement Of Distributed Object Oriented Software Based on Trace Events, European Journal of Scientific Res., 69(4): 527-540.
13. Gyimoty, T., R. Ferenc and I. Siket, 2005. Empirical validation of object - oriented metrics on open source software for fault prediction, IEEE Trans. Software Engineering, 31(10): 897-910.
14. Li Wei and Sallie Henry, 1993. Object-oriented metrics that predict maintainability, Journal of Systems and Software, 23(2): 111-122.
15. Wa, Y. and R. Weber, 1990. An Ontological Model of an Information System, IEEE Transactions on Software Engineering, 16(11): 1282-1292.
16. Yin, Liu Ana and Milanova, 2006. Static analysis for dynamic coupling measures, Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative Research.