International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970)
Volume-1 Number-1 Issue-1 September 2011

A Novel Class, Object and Inheritance based Coupling Measure (COICM) to Find Better OOP Paradigm using JAVA

Narendra Pal Singh Rathore[1], Ravindra Gupta[2]
M.Tech Scholar, Dept. of Computer Science, SSSIST SEHORE, India[1]
Professor in CSE Department at SSSIST Sehore, India[2]

Abstract

The extent of coupling and cohesion in an object-oriented system has implications for its external quality. Various static coupling and cohesion metrics have been proposed and used in past empirical investigations; however none of these have taken the run-time properties of a program into account. As program behavior is a function of its operational environment as well as the complexity of the source code, static metrics may fail to quantify all the underlying dimensions of coupling and cohesion. In this paper we proposed a novel Class, Object and Inheritance based Coupling Measure (COICM) to find better OOP Paradigm using JAVA. By this approach we find the better OOP paradigm. Our Algorithm consist of four phases 1) Authentication 2) Select two Object Oriented Programming Files 3) Count no of Classes, Object and Inheritance 4)Based on the analysis provided in the database we deduce that which programming approach is better in the current situation. Our simulation result shows that it is efficient and applicable on the entire platform. The metric values of class and inheritance diagrams have been compared to prove which concept is good to use and beneficial for developers.

Keywords

OOP, OOA, CBO, Inheritance

1. Introduction

In the process of designing a software system, a software engineer may use a variety of techniques to express and improve the design. In an effort to find some common ground between how a programmer thinks and how a programmer writes code, we focus here only on models used to express the design of a system, either before development or afterwards, as documentation. We concentrate on two models in particular, both of which are regarded as de facto standards: the class diagrams of Unified Modeling Language (UML) [1], and Entity-Relationship diagrams [2].

High quality software design, among many other principles, should obey the principle of low coupling. Stevens et al., who first introduced coupling in the context of structured development techniques, define coupling as "the measure of the strength of association established by a connection from one module to another" [3]. Therefore, the stronger the coupling between modules, i.e., the more inter-related they are, the more difficult these modules are to understand, change, and correct and thus the more complex the resulting software system. Complexity of source code directly relates to cost and quality. Many coupling models are presented in the literature to measure the possible interactions between objects and to measure design complexity. High coupling between objects increases complexity and cost. Low coupling is good for designing object oriented software. Inheritance introduces more interactions among classes [4]. This will increase the complexity. This paper presents a comparison between object oriented interfaces and inheritance class diagrams.

The primary aim of collecting, producing and analyzing software metrics is to provide the capability to predict the future development and maintenance efforts based on past performance [5]. Software metrics are gathered, analyzed, verified and validated at many levels. Metrics are of two types. Traditional metrics are used to measure non object oriented programming and object oriented metrics are used to measure object oriented programming.

In this paper we discuss several aspects of cohesion and coupling on the basis of several aspects. We consider the example of C++ and java for analyzing several aspects of Object Oriented Programming including number of classes and object. We also categorized inheritance behavior on the basis of C++ and java. We also include polymorphic behavior; on the basis of the above criterion we analyze which object oriented programming is good.

The remaining of this paper is organized as follows. We discuss Metric for Class Cohesion in Section 2. In Section 3 we discuss about Object Oriented based coupling. In section 4 we discuss about Recent Scenario. In section 5 we discuss about the proposed method. The conclusions and future directions are given in Section 6. Finally references are given.

2. Metric for Class Cohesion

To describe the cohesion in term of OO paradigm of software development, we first need to understand how the construct of a class stands in OO paradigm. A class may be inherited from zero or more classes in case of Multiple Inheritances in C++ and a class may be derived by zero or more classes. An inherited class is also referred as base class or super class or parent class, whereas derived class is also referred as subclass or child class. Class is composed of its member. Member stands for member attributes and methods of a class. Members of a class may have different access rules, based on the access rules; members can be public, protected and private. Private members cannot be accessed directly from the outside of class; however pointer to private methods or attributes may cause violation to this rule. Protected and public members of inherited class become the part of derived classes and hence such members can be accessed by derived class. In C++, protected members of the class can also be accessed by the friend classes. However, public members of the class can be accessed by all other classes in the system. Good software design dictates that types and methods should have high cohesion and low coupling. High coupling indicates a design that is difficult to reuse and maintain because of its many interdependencies on other types. It is shown in fig 1.

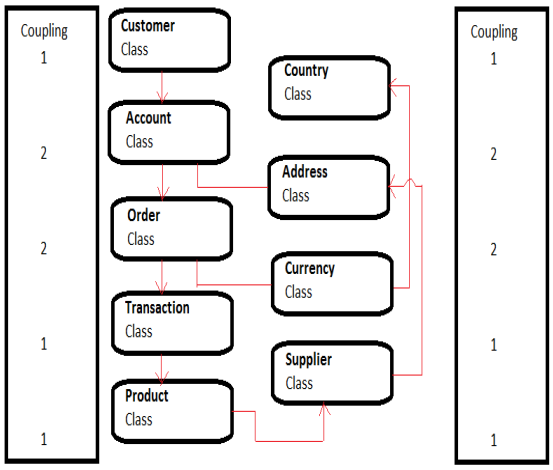

Fig 1 Metric for class cohesion

3. Object Oriented Based Coupling

In OO design, the coupling of a class means the measurement of the interdependence of class with the other classes. In a design of reasonable size design size is ten classes normally classes do not exist in absolute isolation. By going through any OO source code of a working system, one can see that nearly all classes have some kind of relationships with other classes in the design. These relationships, among the classes, create pair-wise interdependencies. Such pair-wise relationships among the classes are the results of design decisions which are made on the specifications of the system.

A good design decision may create a good relationship and a bad design decision may create a bad relation. Here, by good design decision, we mean a decision that makes the OO design easy to reuse, understandable and flexible for modification and adoption in future. Gurp and Bosch, based on an industrial case study, have presented five reasons for software erosion, which revolves around the design decisions made on different stages. During such stages designer team of the system decides which relationship should be used to fulfill the goals (specifications and constraint) of a particular system. Based on the goals of the system and the design skills, a team may design a system that exhibits low or high coupling among the classes.

One way to count the number of class used in this strategy , in this measurement we will count the number of distinct classes with whom a particular class alpha is creating dependency relation. Only one evidence for dependency relation would be enough, caused by any of dependency types to recognize the dependency between two classes. One or more dependency evidences from a class alpha to class beta will increase the counter of this metric by one.

Another approach is to count the total number of evidences including class and inheritance both. This measurement will be used to count total number of evidences for a particular class alpha of 'Used classes by dependency relation'. All types of dependencies will be used to count such evidences. Counter for this measurement will be increased by one with every found evidence for dependency.

High coupling occurs if a class has many links to other classes. Low coupling occurs if a class has zero or few links. Low coupling can be achieved by having less classes linking to one another. For example: You made an AS3 RPG Game using only 1 class (the core engine) that does everything, having all in one will have zero external links to other custom class. Therefore, it is the lowest coupling compare to any OOP (Object Oriented Programming) also; you do not need to bother about coupling issues.

High coupling have their disadvantages as well, as it introduces many linking classes, thus increasing the complexity of managing the project. Furthermore, a class with many links to another class, if 1 class is being taken out, it affects the

other class. Fig2 shows the example of High Coupling pattern.

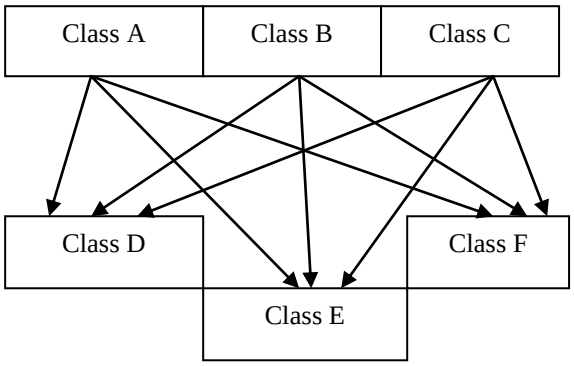

Fig 2 High Coupling Design Pattern

Having a balance coupling (not too high, not too low) is the best, as it allows you to gain both world advantages. The pros of high coupling negate the low coupling cons. The pros of low coupling negate the high coupling cons. To get the right balance coupling, design pattern is introduced. It helps to structure your code to be more reusable, robust and follows object oriented ways. Fig 3 shows the balance coupling pattern.

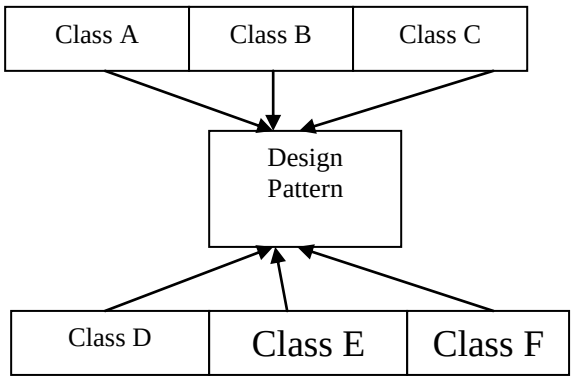

Fig 3 Balanced Coupling pattern

4. Recent Scenario

In 2010, V. Krishnapriya, et al. [6] proposed about the measurement to measure coupling between object (CBO), number of associations between classes (NASSocC), number of dependencies in metric (NDepIN) and number of dependencies out metric (NDepOut) in object oriented programming. A measurement is done for UML class diagrams and interface diagrams. The metric values of class and inheritance diagrams have been compared to prove which concept is good to use and beneficial for developers.

In 2010, Simon Allier et al. [7] express existing definitions of coupling metrics using call graphs. We then compare the results of four different call graph construction algorithms with standard tool implementations of these metrics in an empirical study. Our results show important variations in coupling between standard and call graph-based calculations due to the support of dynamic features.

In 2010, Hongyu Pei Breivold et al. [8] primary studies for this review were identified based on a pre-defined search strategy and a multi-step selection process. Based on their research topics, we have identified four main categories of themes: software trends and patterns, evolution process support, evolvability characteristics addressed in OSS evolution, and examining OSS at software architecture level. A comprehensive overview and synthesis of these categories and related studies is presented as well.

In 2010, Béla Újházi et al. [9] proposed two novel conceptual metrics for measuring coupling and cohesion in software systems. There first metric, Conceptual Coupling between Object classes (CCBO), is based on the well-known CBO coupling metric, while the other metric, Conceptual Lack of Cohesion on Methods (CLCOM5), is based on the LCOM5 cohesion metric. One advantage of the proposed conceptual metrics is that they can be computed in a simpler way as compared to some of the structural metrics.

In 2010, U. L. Kulkarni et al.[10] proposed a case study of applying design measures to assess software quality. Six Java based open source software systems are analyzed using CK metrics suite to find out quality of the system and possible design faults that will reversely affect different quality parameters such as reusability, understandability, testability, maintainability. They also present general guidelines for interpretation of reusability, understandability, testability, maintainability in the context of selected projects.

In 2011, Simon Allier et al. [11] proposed a method to automatically transform an object-oriented application in an operational component-oriented application. They also illustrate this method on a real Java application which is transformed in an operational OSGi application.

5. Proposed Method

In this paper we proposed a novel Class, Object and Inheritance based Coupling Measure (COICM) to find better OOP Paradigm using JAVA. By this approach we find the better OOP paradigm. Our Algorithm consist of four phases 1) Authentication 2)

Select two Object Oriented Programming Files 3) Count no of Classes, Object and Inheritance 4)Based on the analysis provided in the database we deduce that which programming approach is better in the current situation.

In this section we first introduce our algorithm:
Assumption:
Authid-Authenticate User
UnAuthid-Un authorized user
Prog- Different Object Oriented Program
SR- Standard Result
Countcl- no. of classes[Initially 0]
Countob- no. of Object[Initially 0]
Countin-no. of Inheritance[Initially 0]

Algorithm (COICM)

Step 1: [Check Authentication]
{
Enter the userid and password
If(Authid)
{
Print("welcome user");
Go to the COICM Interface
}
Else
Print("Invalid user")

}

Step 2: Select(Prog)

{
Select the Source File
If(.cpp)

[Accept the C++ source file]

Else(.java)
[Accept the .java file]
}

Step 3: Compare (prog)

{
If(.cpp)

[Standard Result CPP]

Else(.java)
[Standard Result java]
}

Step 4: if SR (CPP)
{
Countcl++;
Countob++;
Countin++;
Goto FRC++(Countcl,Countob,Countin);
}

Step 5: if SR (java)
{
Countcl++;
Countob++;
Countin++;
Goto FRjava(Countcl,Countob,Countin);
}

Step 6: if FRC++(Countcl,Countob,Countin)
{
Calculate the result based on different parameter like Generalization,specialization etc.
}

Step 7: if FRjava(Countcl,Countob,Countin)
{
Calculate the result based on different parameter like Generalization, Specialization etc.
}

We can explore the algorithm in the following way:

Step 1: In this section we check proper authentication of the user. If the user is authorized then user enters to the secure zone of COICM Interface. Otherwise first create a valid user and the enter in the secure zone.

Step 2: In this section file selection can be done. We select two object oriented program like C++ and java. Basis on this we can perform several comparison on several parameters.

Step 3: In this section comparison is done. For C++ we use standard result CPP and for java we use standard result java.

Step 4: In this section we count number of classes, object and Inheritance and based on those values we deduce the final result.

Step 5: Final Result is deduced according to the following criterion in the database:

1) Generalization
2) Specialization
3) Association
4) Aggregation
5) Inheritance
6) Polymorphism

Based on the above analysis we can deduce the standard result of C++ program and the java program.

Step 6: Our result set is changed according to the number of classes, object and Inheritance.

Based on the above algorithm we show three screen shot to show the above mechanism.

Fig 4 shows the window where we can count the number of classes, object and inheritance. Based on the parameters we can deduce the final result set. Fig 5 shows the final result based on C++ programs. Fig 6 shows the java result set based on java programs.

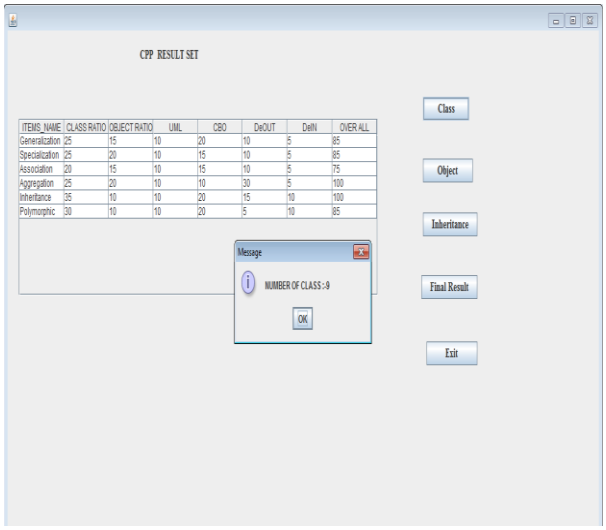

Fig 4 CPP Result Set

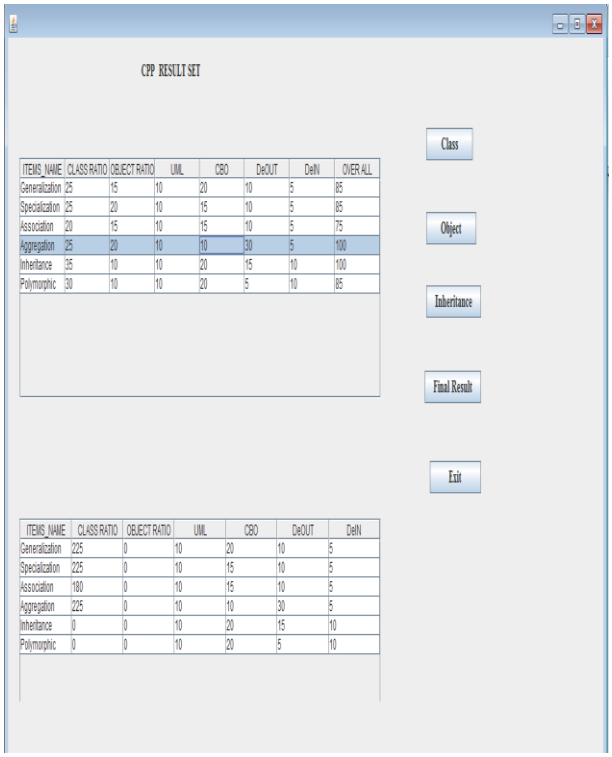

Fig 5 Final Result with CPP file

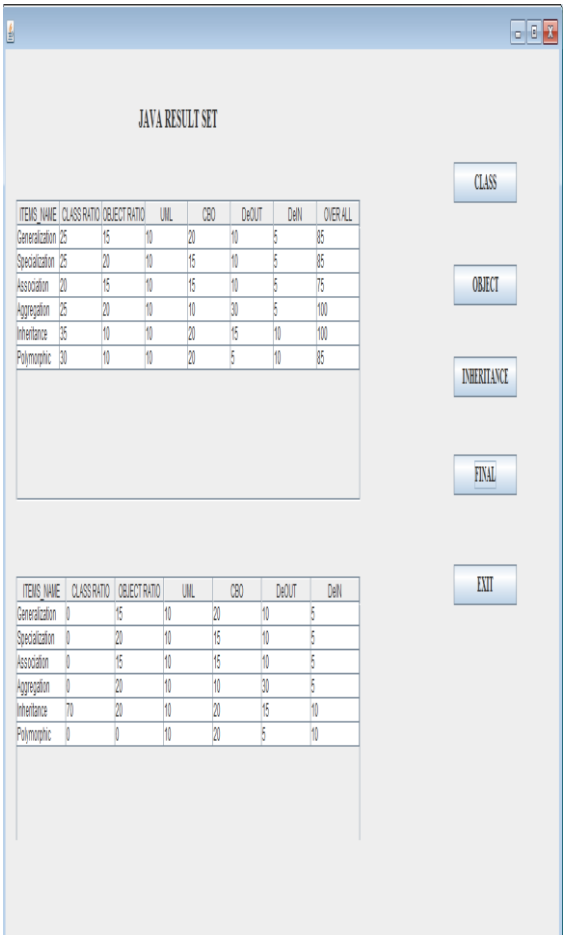

Fig 6 Java Result Set

6. Conclusion and Future Direction

We analyze several aspects about cohesion and coupling. We also analyze several methods. In this paper we proposed a novel Class, Object and Inheritance based Coupling Measure (COICM) to find better OOP Paradigm using JAVA. By this approach we find the better OOP paradigm. Our Algorithm consist of four phases 1) Authentication 2) Select two Object Oriented Programming Files 3) Count no of Classes, Object and Inheritance 4)Based on the analysis provided in the database we deduce that which programming approach is better in the current situation. Our simulation result shows that it is efficient and applicable on the entire platform. The metric values of class and inheritance diagrams have been compared to prove which concept is good to use and beneficial for developers.

References

[1] I. Jacobson, G. Booch, and J. E. Rumbaugh. The unified software development process. Addison-Wesley, 1999.

[2] Database Management Concepts, "Ashutosh Kumar Dubey", S.K Kataria Publication, 2010.
[3] V. Basili, L. Briand, and W. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators," IEEE Trans. Software Eng., vol. 22, no. 10, pp. 751-761, 1996.
[4] Mohsen D. Ghassemi and Ronald R. Mourant,"Evaluation of Coupling in the Context of Java Interfaces", Proceedings OOPSLA 2000.
[5] Christopher L. Brooks, Chrislopher G. Buell, "A Tool for Automatically Gathering Object-Oriented Metrics", IEEE, 1994.
[6] V. Krishnapriya and Dr. K. Ramar, 2010 International Conference on Advances in Computer Engineering,IEEE.
[7] Simon Allier, St´ephane Vaucher, Bruno Dufour, and Houari Sahraoui, 2010 Working Conference on Source Code Analysis and Manipulation, IEEE.
[8] Hongyu Pei Breivold, Muhammad Aufeef Chauhan and Muhammad Ali Babar, 2010 Asia Pacific Software Engineering Conference,IEEE.
[9] Béla Újházi,, Rudolf Ferenc , Denys Poshyvanyk and Tibor Gyimóthy," New Conceptual Coupling and Cohesion Metrics for Object-Oriented Systems", 2010 Working Conference on Source Code Analysis and Manipulation.
[10] U. L. Kulkarni, Y. R. Kalshetty and Vrushali G. Arde," Validation of CK metrics for Object Oriented Design Measurement", IEEE, 2010.
[11] Simon Allier, Salah Sadou, Houari Sahraoui and Regis Fleurquin" From Object-Oriented Applications to ComponentOriented Applications via Component-Oriented Architecture", 2011 Ninth Working IEEE/IFIP Conference on Software Architecture.