Static and Dynamic Coupling and Cohesion Measures in Object Oriented Programming

Vasudha Dixit, Dr. Rajeev Vishwkarma
SVITS Indore
vasu.dixit2007@gmail.com, Rajeev@mail.com

Abstract: ***A large numbers of metrics have been proposed for measuring properties of object-oriented software such as size, inheritance, cohesion and coupling. The coupling metrics presented in this paper exploring the difference between inheritance and interface programming. This paper presents a measurement to measure coupling between object (CBO), number of associations between classes (NASSocC), number of dependencies in metric (NDepIN) and number of dependenciesout metric (NDepOut) in object oriented programming for both static and dynamic analysis. Java programs is used for implementation.In this paper we want to show which concept is good to use and beneficial for software developer***.

Keywords: Measurement, Metrics, Object Oriented programming,Object oriented metrics, *CBO*, Coupling, Inheritance, Interface.

1.Introduction

Object-oriented design and programming is the dominant development paradigm for software systems today. Recently so many languages are object-oriented (OO) programming languages. In object oriented programming we provide abstraction by classes and interfaces. Classes are used to hold functional logic and an interface is used to organize source code. According to object oriented programming, the class provides encapsulation and abstraction and the interface provides abstraction and cannot inherit from one class but can implement multiple interfaces. The above said differences are minor and they are very similar in structure, complexity, readability and maintainability of source code [1][2]. Here, the difference in usage of class inheritance and interface concepts are measured for class diagrams by coupling metrics proposed by Chidamber and Kemrer and Brian. 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 [3]. This will increase the complexity. This paper presents a comparison between object oriented interfaces and inheritance class diagrams.

Software engineering started with a humble beginning and it has slowly come into existence. Now, softwareengineering provides the best solutions for the software problems in object oriented programming. Accurate measurement is a basic necessity for all engineering disciplines and software engineering is not an exception.

To improve software products and process, measurements are essential. Software measurement plays an important role in finding the software quality, performance, maintenance and reliability of software products. The concept of measurement requires appropriate measurement tools to measure, to collect, to verify and validate relevant metric data. Nowadays, many metric tools are available for software measurement [4][5][6]. The main objective of this paper is to measure, analyse and propagate the difference between using object oriented class inheritance and interfaces in class diagrams using coupling measures by metrics. The primary aim of collecting, producing and analysing software metrics is to provide the capability to predict the future development and maintenance efforts based on past performance [4]. Software metrics are gathered, analysed, 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.

2. Literature Survey

Object-oriented measurement has become an increasingly popular research area. Metrics are powerful support tools in software development and maintenance. They are used to assess software quality, to estimate complexity, cost and effort, to control and improve processes. The metrics that are important to calculate reusability are related to inheritance, cohesion and coupling.

2.1. Traditional Metrics

Traditionalmetrics are important to measure non-object oriented programming. Most of the tools are used to gather andanalyse traditional software metrics. Some metrics are very simple and some are more complex. These metrics are also applied to object oriented programming. A large number of object oriented metrics have been developed and also large number of tools exist to collect metrics from programs [4][6] [7].

S No	Metrics	Definition
1	*Source Lines ofCode (SLOC)*	The SLOC metric measuresthe number of physical lines of active code, thatis, no blank orcommented lines code [8]. Counting the SLOCis one of theearliest and easiest approaches to measuringcomplexity. It isalso the most criticized approach [9]. In generalthe higher theSLOC in a module the less understandable andmaintainablethe module is.
2	*McCabeCyclomaticComplexity (CC)*	Cyclomatic complexity is a measure of amodule control flow complexitybased on graph theory [9]. Cyclomaticcomplexity of amodule uses control structures to create acontrol flow matrix,which in turn is used to generate a connectedgraph. Thegraph represents the control paths through themodule. Thecomplexity of the graph is the complexity of themodule [9],10].
3	Commentpercentage(CP)	The CP metric is defined as thenumber of commented lines of code divided bythe number ofnon-blank lines of code. Usually 20% indicatesadequatecommenting for C++ [11]. A high CP valuefacilitates inmaintaining a system.
4	Defect	Number of fault in specification, Design orimplementation.
5	Number ofprocedure	Number of statement groups which can becompiled independently in the program

2.2 Object Oriented Programming and Metrics

Object oriented programming and design are veryimportant in today's environment. It provides generalizedsolutions for many problems in addition to many benefitslike reusability, decomposition of problems into small easilyunderstandable objects and also helps to performmodifications in future and to do functional extensions inalready built systems [12]. Object oriented programming ismore recent and more important in quality softwareprogramming than that of the old style proceduralprogramming. In the last two decades object orientedsoftware engineering receives much attention because objectoriented technology is wide spread [13] [14] [15]. Objectoriented technology and development requires differentapproach to design, implementation and to measure metricscompared to standard set of metrics.A large number of metrics have been developed andproposed by researchers and numerous tools are available tohelp to assess the design, quality, maintenance and to collectmetrics from software programs [6][7] [14][16] [17].Many object oriented metrics in literature lack in theoreticalproof and some have not been validated. The metrics thatevaluate object oriented programming are: classes, methods,inheritance, coupling and cohesion. A method is an operation upon an object and is defined in the class declaration. A message is a request that an object makes of another object to perform an operation. The operation executed as a result of receiving a message is called a method. Cohesion is the degree to which methods within a class are related to one another and work together to provide well-bounded behaviour. Effective object oriented designs maximize cohesion because cohesion promotes encapsulation. Coupling is a measure of the strength of association established by a connection from one entity to another. Classes are coupled when a message is passed between objects; when methods declared in one class use methods or attributes of another class.

3. Class Inheritance and Interfaces

Inheritance is one of the initial features of object oriented programming, and through inheritance a derived class receives the attributes and methods of the base class. The relationship between derived and base class is referred as "is-a/is-a-kind-of". Inheritance feature creates a class hierarchy [18][19].

Nowadays interfaces are heavily used in all disciplines especially in object oriented programming [20]. Withinterface construct, object oriented programming features a good concept with high potential code reusability. Interfaces are used to organize code and provide a solid boundary between the different levels of abstraction [21].It is good to use interfaces in large type of applications because interfaces make the software/program easier toextend, modify and integrate new features. An interface is a prototype for class. With the construct of an interface java allows a concept of high potential for producing a reusable code. Interfaces in object orientedprogramming just contain names and signatures of methods and attributes, but no method implementations. Interfaces are implemented by classes. The inheritance hierarchy of interfaces is independent than that of class inheritance tree. Therefore object oriented languages like java gives higher potential to produce reusable code than abstract classes [22] [23]. Interfaces are prototype for a class. Interfaces can be used like classes in declarations and signatures. With interface construct, object oriented programming features a good concept with high potential code reusability. Today interfaces are heavily used in all disciplines. Interfaces are advisable to be used in large type of applications because the interfaces make the application easier to extend, modify andintegrate new features. Interfaces in object oriented programming just contain names and signatures of methods and attributes, but no method implementations [22]. Interfaces are used to organize code and provide a solid boundary between the different levels of abstraction [21][24].

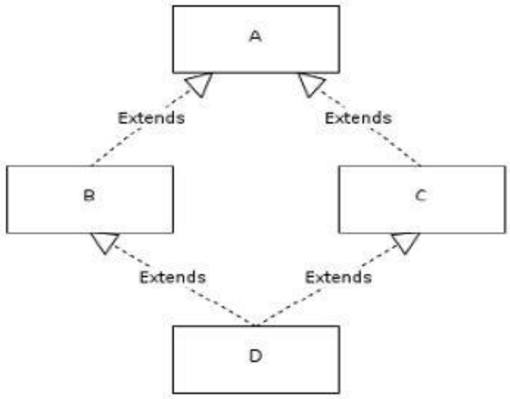


Fig1: Diamond problem of multiple inheritance

The java programming language does not permit multiple inheritance but interfaces provide an alternative. In java a class can inherit from only one class but it can implement more than one interface. Therefore, objects can have multiple types: the type of their own class and the types of all the interfaces that they implement. This means that if a variable is declared to be the type of an interface, its value can reference any object that is instantiated from any class that implements the interface[25].

4. CouplingMetrics [25][26][27][28]

Coupling refers to the degree of independence between parts of the design. To measure coupling in class diagrams there are three types of metrics [26]. In this paper one CK metric is added to measure coupling performance [27]. A measure of coupling is more useful to determine the complexity. The higher the inter object coupling, the more rigorous the testing needs to be. In this paper, four metrics are used to validate the proposed approach.

4.1 Coupling between Objects-CBO

According to CK [28] "CBO for a class is a count of the number of other classes to which it is coupled". A class is coupled with another if the methods of one class use the methods or attributes of the other class. An increase of CBO indicates the reusability of a class will decrease. Multiple accesses to the same class are counted as one access. Only method calls and variable references are counted. Thus, the CBO values for each class should be kept as low as possible [28].

4.2 NASSocC

The number of Association per class metric is defined as the total number of associations a class has with other classes or with itself. This metric is used to measure complexity and coupling [26]. When the number ofassociations are less the coupling between objects are reduced. This metric was introduced by Brian.

4.3 NDepIN

The Number of Dependencies In metric is defined as the number of classes that depend on a given class [26]. This metric is proposed to measure the class complexity due to dependency relationships. The greater the number of classes that depend on a given class, the greater the inter-class dependency and therefore the greater the design complexity of such a class. When the dependencies are reduced the class can function more independently.

4.4 NDepOut

The Number of Dependencies Out metric is defined as the number of classes on which a given class depends. When the metric value is minimum the class can function independently.

5. PROPOSED APPROACH

Goal: Comparing the inheritance and interface concepts in object oriented programming through metrics in both static and dynamic analysis.

Hypothesis: Four object oriented metrics are used for coupling measures in object oriented inheritance classdiagram and interface diagram.

1. Two object oriented programs are used with inheritance concept in this paper.
2. These programs are introduced with maximum possible interfaces.
3. All the four metrics are applied to both inheritance and interface diagrams.
4. The results are compared between inheritance and interface coupling measures.

6. Implementation Results

To validate the above metrics an object oriented inheritance diagram has been taken and possible interfaces have been introduced in class inheritance diagrams. For both inheritance and interface diagrams, all four coupling measures are applied , measured and tabulated with the values based on the above said definitions.

We have used java object oriented programming language for implementation. We have written two programs, one using inheritance and other using interface. The metrics discussed above are applied for both inheritance and interface programs(program1.java and program2.java). The results are show in Table 1.

```
/*******************************************
Program1.java
//Vehicle Classification using Class Inheritance
*******************************************/
class Vehicle
{
public string name;
public int wheelscount;
public void getData()
{
}
public void displayData()
{
}
}
```

```
class LightMotor :Vehicle
{
public int speedlimit;
public int capacity;
}
class HeavyMotor : Vehicle
{
public int speedlimit;
public int capacity;
public string permit;
}

class GearMotor : LightMotor
{
public int gearcount;

}

class NongearMtor : LightMotor
{
}
class Passenger : HeavyMotor
{
public int sitting;
}
class Goods : HeavyMotor
{
}
```

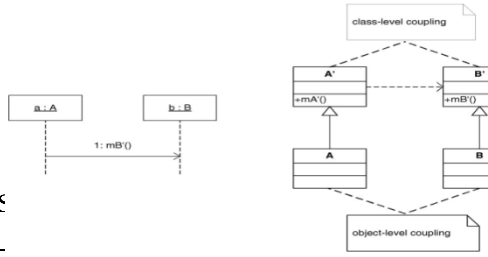
The above said class inheritance in Program1.java is rewritten with possible number of interface in Program2.java

```
/**********************************
Program2.java
 //Vehicle classification using interfaces
**********************************/
interface Lightmotor
{
public void Getspeedcap();
}
interface Vehicle
{
public void GetData();
public void DisplayData();
public void Getnamewc();
}
interface Heavymotor
{
public void Getspeeder();
}
class Gearmotor : Vehicle, Lightmotor
{
public int gearcount;
public void GetData();
public void DisplayData();
public void Getnamewc();
public void Getspeedcap();
}
class Nongearmotor : Vehicle
{
public void GetData();
public void DisplayData();
public void Getnamewc();
}
class Passenger : Vehicle, Heavymotor
{
public void GetData();
public void DisplayData();
public void Getnamewc();
public void Getspeeder();
}
class Goods : Vehicle, Heavymotor
{
public void GetData();
public void DisplayData();
public void Getnamewc();
public void Getspeeder();
}
```

	Metric / Classes	CBO	NASSocC	NDepIN	NDepOut
Program 1	Vehicles	2	2	2	0
	Lightmotor	3	3	2	1
	Heavymotor	3	3	2	1
	Gearmotor	1	1	0	1
	Nongearmotor	1	1	0	1
	Passenger	1	1	0	1
	Goods	1	1	0	1
Program 2	Vehicles	0	0	0	0
	Lightmotor	0	0	0	0
	Heavymotor	0	0	0	0
	Gearmotor	0	0	0	0
	Nongearmotor	0	0	0	0
	Passenger	0	0	0	0
	Goods	0	0	0	0

Class-level versus object-level coupling

Due to inheritance, the class of the object sending or receiving a message may be different from the class implementing the corresponding method. For example, let object a be an instance of class A, which is inherited from ancestor A0. Let A0 implement the method mA0. Let object b be an instance of class B, which is inherited from ancestor B0. Let B0 implement the methodmB0. If object a sends the message mB0 to object b, the message may have been sent from the method source mA0 implemented in class A0 and processed by a method target mB0 implemented in class B0. Thus, in this example, message passing caused two types of coupling: 1) object-level coupling between class A and class B (i.e coupling between instances of A and B) and 2) class level coupling between class A0 and B0. The code may very well show statements where an object of type A invokes from mA0 method mB0 on an object of type B. However, to assume, through static code analysis, that there is class-level coupling between A and B as a result, is simply inaccurate. Both types of coupling, at the class and object levels, need to be captured accurately to address certain applications and must be investigated.



Sequence and Class Diagrams

				DepOut
A’	1	1	1	0
B’	1	1	1	0
A	1	1	0	1
B	1	1	0	1

Dynamic Analysis:

	CBO	NASSocC	DepIN	DepOut
A’	1	1	1	0
B’	1	1	1	0
A	2	1	0	1
B	2	1	0	1

We can see that in case of static analysis two columns (CBO and NASSocC) can be merged but in case of dynamic analysis No of Associations are different from Coupling between Objects. So when we are use Static coupling only 3 metrics are required but when we are use dynamic coupling then 4 metrics are required but in case of dynamic coupling creation the flow of data easily understandable .

7. Conclusion

This paper presents an idea on how to reduce coupling in object oriented programming. Due to the reduction in coupling, developers can produce quality programs. When CBO is reduced reusability will be increased. High coupling will support low encapsulation and produce more faults. Due to the reduction in values of coupling metrics the stability of the structure will be good. When the coupling measures are reduced, the classes can function more independently. The more independent a class it is easier to be reused by another application. To improve modularity and encapsulation the inter object class coupling measures should be minimum. As the number of couples becomes larger the maintenance is more difficult. By using more interfaces compared to inheritance the coupling measures are reduced.

References

i. *V. Krishnapriya, Dr. K. Ramar , Exploring the Difference between Object Oriented Class Inheritance and Interfaces Using Coupling Measures, 2010 International Conference on Advances in Computer Engineering*

ii. *Mathew Cochran,”Coding Better: Using Classes Vs Interfaces”, January 18th, 2009.*

iii. *Mohsen D. Ghassemi and Ronald R. Mourant,”Evaluation of Coupling in the Context of Java Interfaces”, Proceedings OOPSLA 2000, P.No: 47-48, Copyright ACM 2000, 1-58113-307-3/00/10.*

iv. *Christopher L. Brooks, Chrislopher G. Buell, “A Tool for Automatically Gathering Object-Oriented Metrics”, IEEE, 1994.*

v. *Rajib Mall,”Fundamentals of Software Engineering”, Chapter 1,Pg.No:1-18,2nd Edition, April 2004.*

vi. *Rudiger Lincke, Jonas Lundberg and Welf Lowe,”Comparing Software Metrics Tools”, ISSTA’08, July 20-24, 2008, ACM 978-1-59593-904-3/07.*

vii. *Nachiappan Nagappan, Thomas Ball and Andreas Zeller,” Mining Metrics to Predict Component Failures”, Verification and Measurement Group, Microsoft Research, 2005, Redmond, Washington.*

viii. *Lorenz, Mark & Kidd Jeff, Object-Oriented Software Metrics, Prentice Hall, 1994.*

ix. *Tegarden, D., Sheetz, S., Monarchi, D., “Effectiveness of Traditional Software Metrics for Object-Oriented Systems”, Proceedings: 25th Hawaii International Conference on System Sciences, January, 1992, pp. 359-368.*

x. *McCabe and Associates, Using McCabe QA 7.0, 1999, 9861 Broken Land Parkway 4th Floor Columbia, MD 21046.*

xi. *Rosenberg, L., and Hyatt, L., “Software Quality Metrics for Object-Oriented System Environments”, Software Assurance Technology Center, Technical Report SATC-TR-95-1001, NASA Goddard Space Flight Center, Greenbelt, Maryland 20771.*

xii. *Rene Santaolaya Salgado, Olivia G. Fragosco Diaz, Manuel A. Valdes Marrero, Issac M. Vaseuqz Mendez and Shiela L. Delfin Lara, “Object Oriented Metric to Measure the Degree of Dependency Due to Unused Interfaces”, ICCSA 2004, LNCS 3046, P.No: 808-817,2004 @ Springer, Verlag Berlin Heidelberg.*

xiii. *Pradeep Kumar Bhatia, Rajbeer Mann, “ An Approach to Measure Software Reusability of OO Design”, Proceedings of 2nd International Conference on Challenges & Opportunities in Information Technology(COIT-2008),RIMT-IET,Mandi Gobindgarh, March 29,2008.*

xiv. *Santonu Sarkar, Member, IEEE, Avinash C. Kak, and Girish Maskeri Rama,“ Metrics for Measuring the Quality of Modularization of Large-Scale Object-Oriented Software, IEEE Transactions on Software Engineering, Vol. 34, No. 5, Sep-Oct 2008.*

xv. *Terry .C. and Dikel .D.,”Reuse Library Standards Aid Users in Setting up Organizational Reuse Programs”,Embedded System Programming Product News,1996.*

xvi. *El Hachemi Alikacem, Houari A. Sahraoui, “Generic Metric Extraction Framework”, IWSM/ Metrickon, Software Measurement Conference 2006.*

xvii. *Neville I. Churcher, Martin J. Sheppered, ACM Software EngineeringNotes, Vol.20, Issue 2, P.No:69-75, April 1995*

xviii. *Maya Yadav, Pradeep Baniya, Ganesh Wayal, Comparison between Inheritance & interface UML Design through the Coupling Metrics, International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-1, Issue-5, June 2012*

xix. *Ken Pugh,” Interface Oriented Design”, Chapter 5, 2005*

xx. *FriedRich Steimann, Philip Mayer, Andreas MeiBner, "Decoupling Classes with Inferred Interfaces ", Proceedings of 2006 ACM, Symposium on Applied Computing, Pg.No:1404-1408.*

xxi. *Matthew Cochran,"Coding Better: Using Classes Vs. Interfaces", January 18th, 2009.*

xxii. *Markus Mohenen, "Interfaces with Default Implementations in Java", Aachen University of Technology.*

xxiii. *Fried Stiemann, Wolf Siberski and Thomas Kuhne, " Towards the Systematic Use of Interfaces in Java Programming", 2nd Int. Conf. on The Principles and practice of Programming in Java PPJ 2003, P.No 13-17*

xxiv. *Dirk Riehle and Erica Dubach,"Working With Java Interfaces and Classes-How to Separate Interfaces from Implementations", P.No:35-46, Published in Java Report 4, 1999.*

xxv. *Kailash Patidar, RavindraKumar Gupta,Gajendra Singh Chandel , International Journal of Advanced Research in Computer Science and Software Engineering 3(3), March - 2013, pp. 517-521*

xxvi. *Marcela Genero, Mario Piattini and Coral Calero," A Survey of Metrics for UML Class Diagrams", in Journal of Object Technology, Vol. 4, No. 9, Nov-Dec 2005.*

xxvii. *Fried Stiemann, Wolf Siberski and Thomas Kuhne, " Towards the Systematic Use of Interfaces in Java Programming", 2nd Int. Conf. on the Principles and practice of Programming in Java PPJ 2003, P.No:13-17.*

xxviii. *Chidamber S. and Kemerer C.: "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476-493, 1994.*

xxix. *Marcela Genero, Mario Piattini and Coral Calero," A Survey of Metrics for UML Class Diagrams", in Journal of Object Technology, Vol. 4, No. 9, Nov-Dec 2005.*