Package Coupling Measurement in Object-Oriented Software

Article *in* Journal of Computer Science and Technology · March 2009

DOI: 10.1007/s11390-009-9223-6 · Source: DBLP

2 authors:

Varun Gupta
Chandigarh College of Engineering and Technology
33 PUBLICATIONS **867** CITATIONS
SEE PROFILE

Jitender Kumar Chhabra
National Institute of Technology Kurukshetra
133 PUBLICATIONS **2,230** CITATIONS
SEE PROFILE

Gupta V, Chhabra JK. Package coupling measurement in object-oriented software. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 24(2): 273–283 Mar. 2009

Package Coupling Measurement in Object-Oriented Software

Varun Gupta and Jitender Kumar Chhabra

Department of Computer Engineering, National Institute of Technology, Kurukshetra, Kurukshetra-136119, India

E-mail: varun3dec@yahoo.com; jitenderchhabra@rediffmail.com

Received March 16, 2008; revised November 24, 2008.

Abstract The grouping of correlated classes into a package helps in better organization of modern object-oriented software. The quality of such packages needs to be measured so as to estimate their utilization. In this paper, new package coupling metrics are proposed, which also take into consideration the hierarchical structure of packages and direction of connections among package elements. The proposed measures have been validated theoretically as well as empirically using 18 packages taken from two open source software systems. The results obtained from this study show strong correlation between package coupling and understandability of the package which suggests that proposed metrics could be further used to represent other external software quality factors.

Keywords package, coupling, metrics, object-oriented software, understandability

1 Introduction

Ever since the inception of object-oriented programming, class has been the primary unit of organization and reuse. However, over the years it has been acknowledged that a set of collaborating classes is a better unit of organization than a single class. Different names have been given to the set of classes such as "class categories"[1], "clusters"[2], "subject areas"[3], "domains"[4], "subsystems"[5,6], "units"[7]. In UML[8], SmallTalk[9] and Java, these higher order granules are referred to as "packages". This concept of packages is also supported in C++ and C# programming languages in form of namespaces[10]. In all these languages, a package is used to represent a set of classes that might be hierarchically structured and to perform a series of related tasks. For example, in Java, one can write several classes and incorporate them into the same package. Then other Java programs can "import" that package to gain access to the classes contained in the package. Packages can be used to unify important aspects of large system development: system structure, storage management, building, and configuration management. This unification simplifies the developer's task and also provides assurance of reliable and reusable system components. Thus, the importance of packages has been well acknowledged by software community and they have become an integral part of modern object-oriented programming languages such as Java, C#, etc.

Coupling between packages is the manner and degree of interdependence between them. Theoretically, every package is a stand-alone unit, but in reality packages depend on many other packages as either they require services from other packages or provide services to other packages. Thus, coupling between packages cannot be completely avoided but can only be controlled[11]. The coupling between packages is an important factor that affects the quality or other external attributes of software, e.g., reliability, maintainability, reusability, fault-proneness etc. But still, only a few quantitative studies of the actual use of coupling have been conducted at the package level. In this paper, some measures are proposed for measurement of coupling at the package-level in order to achieve good quality software systems. Moreover, it is commonly believed that the early software process phases are the most important ones; since the rest of the development depends on the artifacts they produce. Often, the concepts (e.g., complexity, cohesion, coupling), which are believed to be relevant with respect to code, are also relevant to other artifacts. The measures we propose will be general enough to be applicable to a wide set of artifacts produced in the software process including code and design.

In this paper, we propose metrics for measuring the coupling between packages in a software system. We consolidate our work and validate the metrics suite through Briand's evaluation criteria[12] and by conducting case study on open Java software projects. While defining metrics for measurement of coupling at package

Regular Paper

level, the hierarchical structure of the packages has also been taken into consideration. The key contributions of this paper are the definition and validation (theoretical as well as empirical) of the newly proposed package coupling metrics.

The paper is organized as follows. Section 2 summarizes related work and Section 3 presents basic definitions and properties related to the concept of packages. In Section 4, the package coupling measures are proposed and calculation of these measures has been demonstrated using an example of code written in Java. In Section 5, theoretical validation of the proposed measures is provided, and in Section 6, a case study is conducted using two open source Java projects, and empirical validation of the proposed metrics is performed. Section 7 concludes this paper.

2 Related Work

For object-oriented systems, most of the coupling metrics have been defined up to class level[13−20] and only a few metrics exist for measurement of coupling at the higher levels of abstraction in object-oriented systems[21−26]. Other work related to packages or other higher abstraction levels has been carried out in [27–47].

Lee *et al.* proposed information-based coupling (ICP), a coupling measure for a set of classes based on information flow through method innvocations within classes. They defined coupling of a set of classes as the sum of the couplings of the classes in the set[21]. Using a similar approach, Gui *et al.* proposed system level coupling measure for evaluation of component reusability. They defined coupling of system as mean coupling of all classes present in the system[26].

Martin proposed various package level coupling measures such as afferent coupling, efferent coupling. He defined afferent coupling as the number of classes from other packages that depend on the classes within the package and efferent coupling as the number of classes from other packages that the classes within the package depend upon. In addition, he defined abstractness as the ratio of abstract classes to the total number of classes and instability as the ratio of efferent coupling to total coupling[22,23]. Similarly, Tagoug proposed a coupling measure for subjects. The concept of subjects is quite similar to the packages and he defined subject coupling based on interactions among subjects and also proposed a decomposition quality metric for object-oriented systems on the basis of difference between cohesion and couplings of elements of a system. But, he did not consider the hierarchy of subjects while defining above measures. Also, no justification was provided for exact values assigned to weights of different types of links between elements of a system[24].

Xu *et al.* proposed coupling measures for packages in language Ada95. But the concept of packages in Ada95 is unique to the language and is quite different from the general concept of packages of object-oriented programs[25].

3 Basic Definitions and Properties

In this section, some basic definitions related to the concept of packages are provided and some of the properties regarding package structure of an object-oriented application are discussed, which would be helpful to us in definition of the proposed measures for the package coupling. These definitions and properties are being presented here using formalism based on set theory. This type of formalism helps us to express the proposed measures in consistent and unambiguous manner[12]. In addition, these formal definitions and well-defined properties of the packages would be useful in theoretical validation of the proposed measures.

Package: Packages may consist of classes (and/or interfaces in the case of Java/C# application) and sub-packages as their elements. Further, these sub-packages may contain classes and sub-packages as their elements. This leads to a hierarchical structure of packages in a software system. A package at a hierarchical level i can be represented as

$$p^i = \langle E^{i+1}, R^{i+1} \rangle,$$

where E^{i+1} represents set of elements of package p^i present at level $i+1$, which may be classes or sub-packages, and R^{i+1} is a set of relationships on E^{i+1} at hierarchical level $i+1$, i.e., $R^{i+1} \subseteq E^{i+1} \times E^{i+1}$. The relations on set of elements represent binary directed connections/relationships between pairs of elements of a package.

As shown in Fig.1, for a package at hierarchical level i, p_1^i is represented as $\langle E_1^{i+1}, R_1^{i+1} \rangle$ where E_1^{i+1} represents a set of elements of package p_1^i present at level $i+1$, which are classes (C_1^{i+1}, C_2^{i+1}) and a sub-package (p_2^{i+1}), and R_1^{i+1} is a set of relationships on E_1^{i+1} at hierarchical level $i+1$ which are the relationship between C_1^{i+1} and C_2^{i+1} and relationship between p_2^{i+1} and C_2^{i+1}.

Sub-Package: For any package p^i in a system, subP (p^i) denotes an element of p^i which is package itself and is present at level $i+1$ in the hierarchy. A package $p_1^{i+1} = \langle E_1^{i+2}, R_1^{i+2} \rangle$ is said to be a sub-package of package $p_2^i = \langle E_2^{i+1}, R_2^{i+1} \rangle$ if $p_1^{i+1} \in E_2^{i+1}$, i.e., $p_1^{i+1} = \text{subP}(p_2^i)$. The top level in the package hierarchy in a software system consists of packages, which

are not sub-packages of any other package and the lowest level is occupied by the primitive members, which do not contain any sub-packages.

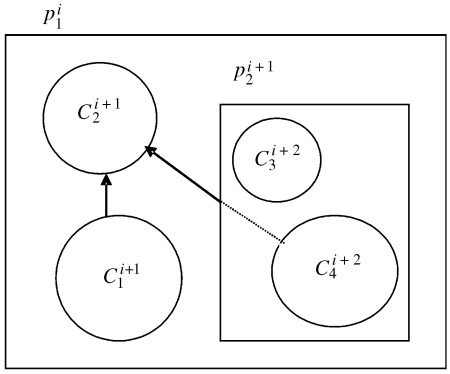

Fig.1. Example depicting packages and their elements.

As shown in Fig.1, p_2^{i+1} is a sub-package of p_1^i as already discussed above and p_2^{i+1} is one of the elements of p_1^i, i.e., $p_2^{i+1} \in E_1^{i+1}$.

Disjoint Packages: All packages of a system are structured into a number of non-overlapping levels, such that for any two packages p_1^i, p_2^i at the same level i, $E_1^{i+1} \cap E_2^{i+1} = \varnothing$. Then, packages p_1^i and p_2^i are said to be disjoint packages. Thus, packages defined at the same hierarchical level are always disjoint. This means, packages defined at the same level never share classes among them. In other words, a single class always belongs to a single package at a particular level of package hierarchy. For instance, if we have two packages p_1^i and p_2^i at the same hierarchical level i and $p_1^i \neq p_2^i$, then $E_1^i \cap E_2^i = \varnothing$.

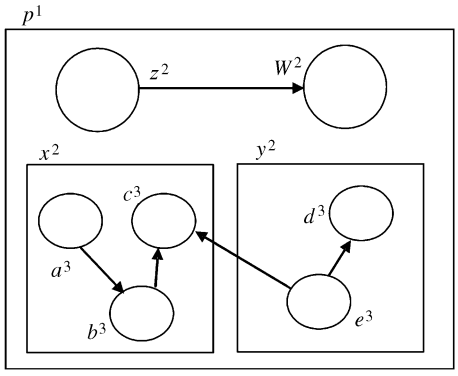

Fig.2. Example depicting disjoint packages.

Fig.2 illustrates the concept of disjoint packages. In this example two packages x^2 and y^2 which are present at the same hierarchical level 2 do not share any classes among them.

Empty Package: A package having no elements and, hence, no relations is termed as empty package. It is denoted by $\langle \varnothing, \varnothing \rangle$.

4 Package Coupling Measurement

As discussed above, packages in an object-oriented system have hierarchical structure and each package can be viewed as a set of elements (classes or sub-packages) and relationships between these elements. These relationships among elements of different packages present at the same hierarchical level give rise to the inter-package coupling. In other words, two different packages present at the same hierarchical level are said to be coupled if the elements of these packages are connected to each other. The elements of the two different packages are said to be connected if there exists a relationship between them and this relationship can be of the type of inheritance relationship, aggregation relationship or simple reference relationship. This connection or relationship between elements of two different packages is denoted by $r(e_i, e_j)$. If there is a connection between two elements e_i and e_j, then, $r(e_i, e_j) = 1$. The presence of connection between two elements can also be represented by $e_i \rightarrow e_j$. Such types of connections always have a direction[48], i.e., if element e_i is connected to element e_j, then it is not necessary that element e_j is also connected to element e_i. Because an element may be dependent on another element, the reverse of this may not be true. For example, if we assume that an element e_1 is dependent on e_2, then it means that any change in implementation of element e_2 is likely to initiate some changes in implementation of element e_1. But any change in implementation of element e_1 will not necessarily demand changes in implementation of e_2. Thus, connection between a pair of elements has to be considered as a directional entity. Thus, connection between a pair of elements is asymmetric in nature, i.e., $r(e_i, e_j) \neq r(e_j, e_i)$. This important aspect of direction of coupling has also been considered at package level in this work. The elements of a package may be classes, interfaces or sub-packages. However, interfaces are unique to C# and Java languages[49] and are quite similar to abstract classes in structure as well as in behavior, as far as inter-package coupling measurement is concerned. Thus, in this study, interfaces would be treated just as a special type of classes. By following the above discussion, the possible types of connections between elements of two different packages are discussed under subsequent sub-headings.

1) *Class−Class Connection*

If both concerned elements of packages are classes (or interfaces), then there exists a class-class type of connection between them and counting such connections

is quite straightforward. These class elements of two packages are said to be connected to each other, if any one of the following relationships exists between them[50]:

• two classes are related by aggregation relationship, i.e., one class has the other class as type of one of its attribute;

• one class inherits another class or one class implements an interface;

• one class has got a method that is invoking method of another class;

• one class has a method referencing an attribute of another class;

• one class has got a method having parameter of the type of another class;

• one class has a method containing a local variable of the type of another class;

• one class has a method invoking a method having a parameter of the type of another class.

This type of connection between the elements of two different packages p_1^i and p_2^i at hierarchical level "i" is represented as:

$$r(e_1^{i+1}, e_2^{i+1}) = 1 | (r(c_1^{i+1}, c_2^{i+1}) = 1 \wedge c_1^{i+1} = e_1^{i+1} \wedge c_2^{i+1} = e_2^{i+1} \wedge e_1^{i+1} \in p_1^i \wedge e_2^{i+1} \in p_2^i).$$

In other words, if there is a connection between a pair of classes present in two different packages at hierarchical level "i", then there exists class-class type of connection between pair of elements of two packages, i.e.,

$$c_1^{i+1} \rightarrow c_2^{i+1} \Rightarrow e_1^{i+1} \rightarrow e_2^{i+1}, \quad \text{where } c_1^{i+1} = e_1^{i+1} \wedge c_2^{i+1} = e_2^{i+1} \wedge e_1^{i+1} \in p_1^i \wedge e_2^{i+1} \in p_2^i.$$

In Fig.3, this type of connection is shown between class *FilledPoly* and class *Poly* as class *FilledPoly* inherits class *Poly*.

2) *Sub-Package−Sub-Package Connection*

Packages may consist of sub-packages as its elements at the next level. Thus, while measuring inter-package coupling, we have to consider connections between pairs of such elements of packages. Such types of connections are recursively defined, as one sub-package is said to be connected to another sub-package if elements of one sub-package have got one or more relationships with the elements of another sub-package. This type of connection between elements of two different packages at hierarchical level "i" is denoted by:

$$r(e_1^{i+1}, e_2^{i+1}) = 1 | (r(c_1^{\geqslant i+2}, c_2^{\geqslant i+2}) = 1 \wedge c_1^{\geqslant i+2} \in e_1^{i+1} \wedge c_2^{\geqslant i+2} \in e_2^{i+1} \wedge e_1^{i+1} = subP(p_1^i) \wedge e_2^{i+1} = subP(p_2^i)).$$

It can also be stated that if elements (present at $i+1$ or higher levels) of sub-packages of two different packages are connected, then these sub-package elements of two packages are also connected, i.e.,

$$c_1^{\geqslant i+2} \rightarrow c_2^{\geqslant i+2} \Rightarrow e_1^{i+1} \rightarrow e_2^{i+1}, \quad \text{where } (c_1^{\geqslant i+2} \in e_1^{i+1} \wedge c_2^{\geqslant i+2} \in e_2^{i+1} \wedge e_1^{i+1} = subP(p_1^i) \wedge e_2^{i+1} = subP(p_2^i)).$$

As shown in Fig 3., this type of connection exists between packages *shape.filled.fillR*ect and *shape.empty.rect.* due to the relationship between class *FilledRectangle* which is present in package *shape.filled.fillR*ect and class *Rectangle* which is contained in package *shape.empty.rect.*

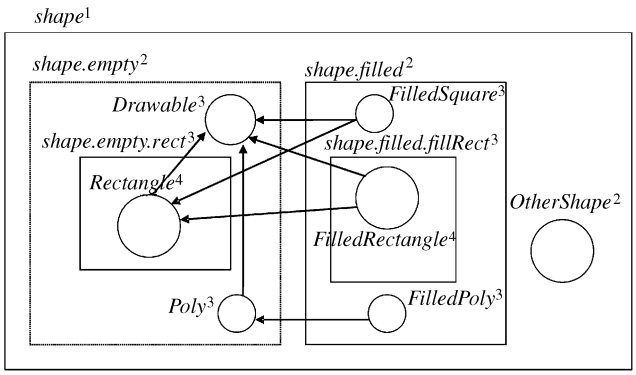

Fig.3. Example of package structure implemented in Java.

3) *Sub-Package−Class Connection*

This type of connection exists between elements of a pair of packages such that the element of the first package is of sub-package type and the element of the second package is of class type. A sub-package element of a package is said to be connected to a class element of another package present at hierarchical level i, if any class element (present at $i+2$ or higher levels) of sub-package present at level $i+1$ has got a relationship with the class element present at level $i+1$ of the other package. This type of connection can be specified as:

$$r(e_1^{i+1}, e_2^{i+1}) = 1 | (r(c_1^{\geqslant i+2}, c_2^{i+1}) = 1 \wedge c_1^{\geqslant i+2} \in e_1^{i+1} \wedge c_2^{i+1} = e_2^{i+1} \wedge e_1^{i+1} = subP(p_1^i) \wedge e_2^{i+1} \in p_2^i).$$

Alternatively, this type of connection can be represented as:

$$c_1^{\geqslant i+2} \rightarrow c_2^{i+1} \Rightarrow e_1^{i+1} \rightarrow e_2^{i+1}, \quad \text{where } (c_1^{\geqslant i+2} \in e_1^{i+1} \wedge c_2^{i+1} = e_2^{i+1} \wedge e_1^{i+1} = subP(p_1^i) \wedge e_2^{i+1} \in p_2^i).$$

This type of connection is shown in Fig.3, between

package *shape.filled.fillRect* and *Drawable* interface due to the fact that class *FilledRectangle* present in package *shape.filled.fillRect* implements *Drawable* interface.

4) *Class−Sub-Package Connection*

Class−sub-package connection is supposed to exist between a class element of one package and a sub-package element of another package. The class of one package is said to be connected to the sub-package of the package, if there is a connection from the class-element at level $i+1$ of the first package to any class element (present at $i+2$ or higher levels) of the sub-package of the second package. This connection can be formally written as:

$$r(e_1^{i+1}, e_2^{i+1}) = 1 | (r(c_1^{i+1}, c_2^{\geqslant i+2}) = 1 \wedge c_1^{i+1} = e_1^{i+1} \wedge c_2^{\geqslant i+2} \in e_2^{i+1} \wedge e_1^{i+1} \in p_1^i \wedge e_2^{i+1} = subP(p_2^i)).$$

This type of connection can also be given as:

$$c_1^{i+1} \rightarrow c_2^{\geqslant i+2} \Rightarrow e_1^{i+1} \rightarrow e_2^{i+1}, \quad \text{where } (c_1^{i+1} = e_1^{i+1} \wedge c_2^{\geqslant i+2} \in e_2^{i+1} \wedge e_1^{i+1} \in p_1^i \wedge e_1^{i+1} \in p_1^i \wedge e_2^{i+1} = subP(p_2^i)).$$

Fig.3 shows this type of connection between class *Filled-Square* and package *shape.empty.rect* because of relationship between class *FilledSquare* and class *Rectangle* present in sub-package *shape.empty.rect*.

Coupling between two packages is the total number of directed connections between ordered and unique pairs of their elements which can be classes or sub-packages. The presence of different types of elements leads to different types of connections between elements as defined above, i.e., class−class, sub-package−sub-package, class−sub-package, sub-package−class. These connections are directed connections. Thus, coupling between a pair of packages p_a^i and p_b^i present at hierarchical level i having n elements and m elements respectively is defined as:

$$Coup(p_a^i, p_b^i) = \begin{cases} 0, & (n = 0 \text{ or } m = 0); \\ \sum_{x=1}^{n} \sum_{y=1}^{m} r(e_x^{i+1}, e_y^{i+1}) & \\ \quad + \sum_{y=1}^{m} \sum_{x=1}^{n} r(e_y^{i+1}, e_x^{i+1}), & \\ \quad (n \geqslant 1 \wedge m \geqslant 1). & \end{cases}$$

Here, n is the number of elements (classes or sub-packages) at level $i+1$ in a package p_a^i, and m is the number of elements (classes or sub-packages) at level $i+1$ in a package p_b^i.

Also, $e_x^{i+1} \in p_a^i$ and $e_y^{i+1} \in p_b^i$, i.e., e_x^{i+1} represents an element present at hierarchical level $i+1$ in a package p_a^i, and e_y^{i+1} represents an element present at hierarchical level $i+1$ in a package p_b^i. The connection from an element of packages p_a^i to an element of p_b^i at level $i+1$ is denoted by $r(e_x^{i+1}, e_y^{i+1})$, and a connection from an element of packages p_b^i to an element of p_a^i at level $i+1$ is denoted by $r(e_y^{i+1}, e_x^{i+1})$. $r(e_x^{i+1}, e_y^{i+1})$ can be defined as follows:

$$r(e_x^{i+1}, e_y^{i+1}) = \begin{cases} 1, & e_x^{i+1} \rightarrow e_y^{i+1}; \\ 0, & \text{otherwise}; \end{cases}$$

$r(e_x^{i+1}, e_y^{i+1})$ and $r(e_y^{i+1}, e_x^{i+1})$ represent binary directed connection between elements of a pair of packages. Two elements belonging to different packages are said to be connected only if there exists a connection between them. These connections can be any one of the types explained above.

$\sum_{x=1}^{n} \sum_{y=1}^{m} r(e_x^{i+1}, e_y^{i+1}) + \sum_{y=1}^{m} \sum_{x=1}^{n} r(e_y^{i+1}, e_x^{i+1})$ represents the actual number of connections between all ordered and unique pairs of elements present at hierarchical level $i+1$ of two packages, and also in accordance with our above discussions, $r(e_x^{i+1}, e_y^{i+1})$ is not equal to $r(e_y^{i+1}, e_x^{i+1})$.

Special Cases

If $n = 0$, there is no element in the first package and no connection is possible from or to this package. Thus, the coupling of this package with any other package is zero, and if $m = 0$, it means the second package contains no element and the number of connections to or from this package is zero. Thus, the coupling is nil between such packages.

The total coupling of a package at a hierarchical level can be defined as the summation of coupling of the package with all other packages present at the same hierarchical level, i.e., Package coupling metric (PCM) for a package p_a^i at hierarchical level "i" is represented as:

$$PCM(p_a^i) = \sum_{b=1 \wedge b \neq a}^{t} Coup(p_a^i, p_b^i)$$

where $Coup(p_a^i, p_b^i)$ is defined as above, i.e., coupling between a pair of packages at hierarchical level i and t is the total number of packages present at hierarchical level i.

4.1 Demonstration of Package Coupling Measurement

The graph shown in Fig.3 represents the package structure as defined by example of Java code given below. This graph does not follow any standard conventions and is drawn only to demonstrate the package

level coupling measurement. This graph shows packages and sub-packages in rectangular shapes and classes and interfaces in round shapes. This graph shows nested package structure of Java programs. A package may contain other packages as its elements, which can be further nested. The relationships between elements are shown as directed arrows from one element to the other in the graph.

An example of code written in Java using packages (only for demonstration purpose) is given below.

```
// Java Example
package shape;
public class OtherShape {
    public void draw(Graphics g) {
        ... // do something
    }
    ... //other methods and variables
}
package shape.empty;
public interface Drawable {
    public void draw(Graphics g);
}
Public class Poly implements Drawable {
    public void draw(Graphics g) {
        ...//do something
        draw a Polygon    }
    ...//other methods and variables
}
package shape.empty.rect;
import shape.empty.Drawable;
public class Rectangle implements Drawable {
    public void draw(Graphics g) {
        ...//do something
        //draw a Rectangle    }
    ...//other methods and variables
}
package shape.filled;
import shape.empty.*;
import shape.empty.rect.Rectangle;
public class FilledSquare extends Rectangle implements
     Drawable {
        public void draw(Graphics g) {
          ... //do something — draw a filled Square
        }
        void findArea( ) {
        ... //find area of filled Square    }
        ... //other methods and variables
        }
public class FilledPoly extends Poly {
        public void draw(Graphics g) {
          ... //do something — draw a filled Square
          }
          void findArea( ) {
          ... //find area of filled Poly
          }
          ... //other methods and variables
}
package shape.filled.fillRect;
import shape.empty.*;
import shape.empty.rect.Rectangle;
public class FilledRectangle extends Rectangle imple-
ments Drawable {
        public void draw(Graphics g) {
        ... //do something — draw a filled Rectangle
        }
    void findArea( ) {
        ... //find area of filled Rectangle   }
          ... //other methods and variables
}
```

At top level, package *shape* contains three elements: one class *OtherShape* and two packages *shape.empty* and *shape.filled*. Further package *shape.empty* contains one interface *Drawable*, one class *Poly* and one sub-package *shape.empty.rect*, which further contains one class *Rectangle* as its element. An other package *shape.filled* contains two classes *FilledSquare* and *FilledPoly* and sub-package *shape.filled.fillRect*, further contains one class *FilledRectangle* as its element. Here, we will demonstrate the calculation process of our proposed measure. This measure will be used to calculate coupling between two packages *shape.filled* and *shape.empty*, since, the number of elements in *shape.filled* is 3 and number of elements in *shape.empty* is also 3.

Hence, $n = 3$ and $m = 3$.

Connections present between packages *shape.filled* and *shape.empty* are explained below:

Class−Class Connection: class *FilledPoly* extends class *Poly* and *FilledSquare* implements *Drawable*.

Sub-package−Sub-package Connection: due to the relation between class *FilledRectangle* present in sub-package *shape.filled.fillRect* and class *Rectangle* present in sub-package *shape.empty.rect*.

Sub-package−Class Connection: due to the fact that class *FilledRectangle* present in sub-package *shape.filled.fillRect* implements *Drawable* interface.

Class−Sub-package Connection: due to the relation between class *FilledSquare* and class *Rectangle* present in sub-package *shape.empty.rect*.

Here, $\sum_{x=1}^{n}\sum_{y=1}^{m} r(e_x^{i+1}, e_y^{i+1}) + \sum_{y=1}^{m}\sum_{x=1}^{n} r(e_y^{i+1}, e_x^{i+1}) = 5$.

Thus, $Coup(shape.empty^2,\ shape.filled^2) = 5$ and Package Coupling Metric, $PCM(shape.filled^2) = 5$.

5 Theoretical Validation

The proposed inter-package coupling measures are validated theoretically by analyzing their mathematical

properties. For this purpose, five properties given by Briand *et al.* in [12] are used and these properties provide a useful guideline in construction and validation of coupling measures in a precise manner and these properties are necessary to prove the usefulness of a coupling measure although not completely sufficient.

Property 1: *Non-Negativity*

The value of coupling between a pair of packages and total coupling of a package with other packages at the same hierarchical level as defined by our measures in an object-oriented system will always be non-negative,

$$Coup(p_a^i, p_b^i) \geqslant 0 \text{ and } PCM(p_a^i) \geqslant 0.$$

Thus, our proposed measures satisfy Property 1.

Property 2: *Null Value*

If the number of elements in one of the two packages is zero or there is no relationship between two packages, then coupling will be nil between these packages and also if a package has got no relationship with other packages present at the same level of hierarchy, then the value of PCM will be nil for this package.

If $n = 0$ or $m = 0$ or

$$\sum_{x=1}^{n}\sum_{y=1}^{m} r(e_x^{i+1}, e_y^{i+1}) + \sum_{y=1}^{m}\sum_{x=1}^{n} r(e_y^{i+1}, e_x^{i+1}) = 0$$
$$\Rightarrow\ Coup(p_a^i, p_b^i) = 0 \text{ and } PCM(p_a^i) = 0.$$

Since the proposed measures have got a null value in a well definition situation, our measures satisfy Property 2.

Property 3: *Monotonicity*

If an additional relationship is added between a pair of packages whose inter-package coupling is to be measured, then according to this property the coupling between the packages must not decrease and similarly, for an addition of a relationship of a package with any other package on the same level, total coupling of the package must not decrease.

According to the definition of our measures, coupling between a pair of packages is the count of relationships among their elements. If we add an additional relationship between them, then the coupling between them can only increase or at least remain the same, but can never decrease in any case. Similarly, if a relationship is added between a package and any other package present at the same level. Then, the amount of coupling for that can only increase or at least remain the same but can never decrease. The coupling between a pair of packages at level i is given by $Coup(p_a^i, p_b^i)$ and after addition of another relationship between packages, let coupling between them be $Coup'(p_a^i, p_b^i)$. Similarly, the coupling for a package at level "i" is represented by $PCM(p_a^i)$ and after addition of another relationship between this package and other packages, let the coupling of the package be $PCM'(p_a^i)$. Then, in any case,

$$Coup'(p_a^i, p_b^i) \geqslant Coup(p_a^i, p_b^i) \text{ and } PCM'(p_a^i) \geqslant PCM(p_a^i).$$

Thus, our proposed measures satisfy Property 3 also.

Property 4: *Merging of Packages*

This property states that merging of two packages must not increase coupling of resulting system because some of the relationships may disappear on merger.

Let P be an object-oriented system, and p_a^i, $p_b^i \in P$ be two packages in P. Let p^i be the package which is obtained by merging of p_a^i and p_b^i. Let P' be the new object-oriented system which is identical to P except that packages p_a^i and p_b^i are replaced by p^i. Then, in any case, $Coup(P) \geqslant Coup(P')$. Thus, our proposed measures satisfy Property 4.

Property 5: *Merging of Unconnected Packages*

This property states that merging of two unconnected packages must not increase coupling of resulting system. When two or more packages having no relationships between them are merged, coupling cannot increase because apparently unconnected packages are being encapsulated together in a single package. Let p_a^i and p_b^i be two packages in an object-oriented system P. Let p^i be the package, which is the union of p_a and p_b. Let P' be the new object-oriented system, which is identical to P except that packages p_a^i and p_b^i are replaced by p^i. If no relationships exist between packages p_a^i and p_b^i, then

$$Coup(P) \geqslant Coup(P').$$

Thus, this property is well satisfied by our proposed measures.

6 Case Study on Open-Source Java Projects

Open source software projects are extensively available on the web. These open source software are freely available for download and are very useful for the purpose of research. The open source software projects chosen for this case-study are XGen Source Code Generator[51] and Jakarta Element Construction Set (ECS)[52]. The XGen software is available from website sourceforge.net and is distributed under General Public License (GPL). This software creates Java source code from a simple XML document and its main function is to generate JDBC compliant beans that allow object level persistence to relational databases. It has full support for all JDBC 2.0 data types, including BLOB and CLOB. Whereas, Jakarta ECS can be

downloaded from website jakarta.apache.org. This software is a Java API for generating elements for various markup languages such as HTML 4.0 and XML.

Table 1. Information Regarding Projects Taken for Case Study

Software Project	XGen	Jakarta ECS
No. Packages	6	12
Total No. Classes	52	418

The basic data about these two projects are given in Table 1. The six packages having 52 classes are taken from XGen project and twelve packages having 418 classes are taken from Jakarta ECS project. Table 2 lists the names of packages of XGen and the number of classes contained in each package.

Table 2. Six Packages Taken from XGen

Sr. No.	Package Name	Total No. Classes
1	workzen.xgen.ant	5
2	workzen.xgen.engine	2
3	workzen.xgen.loader	7
4	workzen.xgen.model	17
5	workzen.xgen.test	17
6	workzen.xgen.util	4

Similarly, Table 3 lists the names of packages of Jakarta ECS and the number of classes contained in each package.

Table 3. Twelve Packages Taken from Jakrata ECS

Sr. No.	Package Name	Total No. Classes
1	org.apache.ecs.examples	1
2	org.apache.ecs.factory	1
3	org.apache.ecs.filter	5
4	org.apache.ecs.html	91
5	org.apache.ecs.html2ecs	1
6	org.apache.ecs.jsp	15
7	org.apache.ecs.rtf	46
8	org.apache.ecs.storage	3
9	org.apache.ecs.vxml	52
10	org.apache.ecs.wml	45
11	org.apache.ecs.xhtml	91
12	org.apache.ecs.xml	3

6.1 Results and Discussions

We have applied our proposed Package Coupling Metrics to eighteen packages taken from XGen and ECS software systems. Table 4 summarizes the package coupling observed in these eighteen packages with their respective number of classes contained in them. The Total Package Coupling for a package is measured as the summation of coupling of the package with all other packages present at the same hierarchical level in the software system. Fig.4 shows the number of classes and the corresponding total package coupling for all 18 packages. Although, a package having the large number of classes is likely to have more connections with other packages, but this may not be always true. For example, package org.apache.ecs.vxml has got 52 classes and amount of package coupling as 2 whereas package org.apache.ecs.wml has got 45 classes and amount of package coupling as 70. From Fig.4, it can be easily observed that there is no definite correlation between the number of classes and the amount of package coupling for a package.

Table 4. Package Coupling Measurement

Sr. No.	Package Name	Total No. Classes	Package Coupling Metric (PCM)
1	workzen.xgen.ant	5	6
2	workzen.xgen.engine	2	5
3	workzen.xgen.loader	7	39
4	workzen.xgen.model	17	35
5	workzen.xgen.test	17	28
6	workzen.xgen.util	4	10
7	org.apache.ecs.examples	1	2
8	org.apache.ecs.factory	1	0
9	org.apache.ecs.filter	5	6
10	org.apache.ecs.html	91	120
11	org.apache.ecs.html2ecs	1	2
12	org.apache.ecs.jsp	15	1
13	org.apache.ecs.rtf	46	5
14	org.apache.ecs.storage	3	0
15	org.apache.ecs.vxml	52	2
16	org.apache.ecs.wml	45	70
17	org.apache.ecs.xhtml	91	130
18	org.apache.ecs.xml	3	12

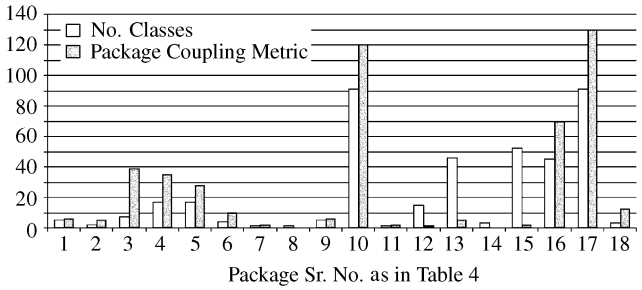

Fig.4. Package coupling metric and No. classes in each package.

6.2 Empirical Validation

In this subsection, we validate the proposed Package Coupling Metrics empirically. The proposed metrics are

internal attributes of a software system, for the purpose of proving the usefulness of these Metrics, these metrics should be correlated to some external quality attributes of the software system. In this study, we correlate the proposed Package Coupling Metrics with the external quality factor, understandability of the software system. For this purpose, we assigned these 18 packages to three teams and each team having three members. These members of all the teams have almost similar experience of Java programming. All the teams are required to fully understand the functionality of these packages and assess the effort required to understand each package and rank the effort from 1 to 10. A higher rank indicates that more effort spent on understanding the package.

Table 5 shows the ranked efforts to understand the 18 packages by the three teams. The average effort required to understand a package is also given in Table 5.

Table 5. Ranks Assigned for Understandability of Packages

Sr. No.	Package Name	Team			Average Effort
		1	2	3	
1	workzen.xgen.ant	3	1	2	2.0
2	workzen.xgen.engine	2	2	3	2.3
3	workzen.xgen.loader	5	5	6	5.3
4	workzen.xgen.model	6	5	7	6.0
5	workzen.xgen.test	5	4	4	4.3
6	workzen.xgen.util	2	3	3	2.6
7	org.apache.ecs.examples	2	1	1	1.3
8	org.apache.ecs.factory	1	1	1	1.0
9	org.apache.ecs.filter	3	2	1	2.0
10	org.apache.ecs.html	8	9	10	9.0
11	org.apache.ecs.html2ecs	1	1	2	1.3
12	org.apache.ecs.jsp	2	1	2	1.6
13	org.apache.ecs.rtf	2	2	3	2.3
14	org.apache.ecs.storage	2	1	3	2.0
15	org.apache.ecs.vxml	1	1	2	1.3
16	org.apache.ecs.wml	7	8	9	8.0
17	org.apache.ecs.xhtml	8	9	10	9.0
18	org.apache.ecs.xml	4	3	5	4.0

As discussed above, we expect some correlation between the amount of package coupling and the effort required to understand the package. In order to check this correlation, we test the following null hypothesis.

H_0: *There is no correlation between the Package Coupling Metric and the effort required to understand the package.*

To test this hypothesis, we need to calculate the amount of correlation between two variables Package Coupling Metric and Average Effort required for understanding the package. For this purpose, here we used Spearman's rank correlation method[53]. The results obtained by applying this test are given in Table 6.

Table 6. Spearman's Correlation Test Results

Variables	PCM & Average Effort
Correlation Coefficient (r)	0.73
Significance Level (p)	0.05

The correlation coefficient between two variables Package Coupling Metric and Average Effort comes out to be 0.73 at 0.05 significance levels. Thus, we reject the null hypothesis H_0 and conclude that there is strong correlation between package coupling and effort required to understand the package, i.e., understandability of the package. This strong correlation between Package Coupling Metric and understandability of the package provides the empirical evidence that proposed metrics are valid indicators of external quality factors of the software systems.

7 Conclusion

In this paper, we have proposed new metrics for measurement of package level coupling based on formal definitions and properties of the packages. The package represents unification of classes and sub-packages and the proposed metrics take into account different types of connections between two different packages such as class−class, sub-package−sub-package, sub-package−class and class−sub-package. The hierarchical structure of packages and direction of connection between packages have also been taken into consideration during the measurement of package level coupling. The proposed metrics have been validated theoretically as well as empirically. The empirical validation of the proposed metrics has been provided by evaluating two open source Java projects XGen and Jakarta ECS, which consist of 18 packages, which consist of 470 classes. Our study clearly reflect that the proposed metrics are valid indicators of external quality attributes of the software such as understandability. We believe that this work will encourage other researchers and developers to use the results obtained from this study to predict and measure several other software quality attributes.

References

[1] Booch G. Object Oriented Analysis and Design with Applications. 2nd Ed., Redwood City: Benjamin Cummings Publishing Company Inc., LA, USA, 1994.

[2] Meyer B. Object Success — A Manager's Guide to Object Orientation, Its Impact on the Corporation, and Its Use for Reengineering the Software Process. Prentice Hall, 1995.

[3] Coad P, Yourdon E. Object-Oriented Analysis. Prentice Hall, 1991.

[4] Page-Jones M. What Every Programmer Should Know about Object-Oriented Design. Dorset House New York, USA, 1995.
[5] Rumbaugh J, Blaha M, Premerlani W, Eddy F, Lorensen W. Object-Oriented Modeling and Design. Prentice Hall, Englewood Cliffs, NJ, USA, 1991.
[6] Jacobson I, Christerson M, Jonsson P, Övergaard D G. Object-Oriented Software Engineering — A Use Case Driven Approach. ACM Press, Addison-Wesley, MA, USA, 1992.
[7] Cantu M. Mastering Delphi 2 for Windows 95/NT. Sybex, 1996.
[8] Booch G, Jacobson I, Rumbaugh J. UML Semantics. Rational Software Corporation, Version 1.0, 1997.
[9] Lewis S. The Art and Science of Smalltalk. Prentice-Hall, 1995.
[10] Stroustrup B. The C++ Programming Language. Third Edition, Addison-Wesley Publishing Company, Massachusetts, USA, 1997.
[11] DeMarco T. Controlling Software Projects. Yourdon Press, New York, USA, 1982.
[12] Briand L, Morasca S, Basili V. Property-based software engineering measurement. *IEEE Transactions of Software Engineering*, 1996, 22(1): 68–86.
[13] Chidamber S R, Kemerer C F. Towards a metrics suite for object oriented design. In *Proc. the 6th ACM Conf. Object-Oriented Programming: Systems, Languages and Applications* (*OOPSLA*), Phoenix, AZ, Oct. 6–11, 1991, pp.197–211.
[14] Chidamber S R, Kemerer C F. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 1994, 20(6): 476–493.
[15] Eder J, Kappel G, Schrefl M. Coupling and cohesion in object-oriented systems. Technical Report, University of Klagenfurt, 1994.
[16] Churcher N I, Shepperd M J. Comments on 'A metrics suite for object-oriented design'. *IEEE Transactions of Software Engineering*, 1995, 21(3): 263–265.
[17] Li W, Henry S. Object-oriented metrics that predict maintainability. *Journal of Systems and Software*, 1993, 23(2): 111–122.
[18] Li W, Henry S, Kafura D, Schulman R. Measuring object-oriented design. *Journal of Object-Oriented Programming*, 1995, 8(4): 48–55.
[19] Hitz M, Montazeri B. Measuring coupling in object-oriented systems. *Object Currents*, 1996, 1(4): 124–136.
[20] Briand L, Devanbu P, Melo W. An investigation into coupling measures for C++. In *Proc. 19th Int. Conf. Software Eng.*, Boston, May 17–23, 1997, pp.412–421.
[21] Lee Y S, Liang B S, Wu S F, Wang F J. Measuring the coupling and cohesion of an object-oriented program based on information flow. In *Proc. International Conference on Software Quality*, Maribor, Slovenia, Nov. 6–9, 1995, pp.81–90.
[22] Martin R. Object oriented design quality metrics: An analysis of dependencies. *ROAD*, 1995, 2(3).
[23] Martin R. Agile Software Development, Principles, Patterns, and Practices. Prentice Hall, 2002.
[24] Tagoug N. Object-oriented system decomposition quality. In *Proc. the 7th IEEE International Symposium on High Assurance Systems Engineering*, Tokyo, Japan, Oct. 23–25, 2002, pp.230–235.
[25] Xu B, Chen Z, Zhao J. Measuring cohesion of packages in Ada95. In *Proc. Annual ACM SIGAda International Conference on Ada: The Engineering of Correct and Reliable Software for Real-Time & Distributed Systems Using Ada and Related Technologies*, San Diego, California, USA, Dec. 7–11, 2003, pp.62–67.
[26] Gui G, Scott P D. Coupling and cohesion measures for evaluation of component reusability. In *Proc. International Workshop on Mining Software Repositories*, Shanghai, China, May 22–23, 2006, pp.18–21.
[27] Yu L G, Ramaswamy S. Component dependency in object-oriented software. *Journal of Computer Science and Technology*, May 2007, 22(3): 379–386.
[28] Xu T, Qian K, He X. Service oriented dynamic decoupling metrics. In *Proc. the 2006 International Conference on Semantic Web and Web Services* (*SWWS'06*), Las Vegas, USA, June 26–29, 2006, pp.170–176.
[29] Washizaki H, Yamamoto H, Fukazawa Y. A metrics suite for measuring reusability of software components. In *Proc. the Ninth International Software Metrics Symposium* (*METRICS'03*), 2003.
[30] Narasimhan V L, Hendradjaya B. Some theoretical considerations for a suite of metrics for the integration of software components. *Information Sciences*, 2007, 177: 844–864.
[31] Seng O, Bauer M, Biehl M, Pache G. Search-based improvement of subsystem decompositions. In *Proc. GECCO'05*, Washington, DC, USA, June 25–29, 2005, pp.1045–1051.
[32] Franch X, Carvallo J P. A quality-model-based approach for describing and evaluating software packages. In *Proc. IEEE Joint International Conference on Requirements Engineering* (*RE'02*), Essan, Germany, Sept. 9–13, 2002, pp.1–8.
[33] Ponisio L, Nierstrasz O. Using contextual information to assess package cohesion. Technical Report No. IAM-06-002, 2006, Institute of Applied Mathematics and Computer Sciences, University of Berne, 2006.
[34] Lieberherr K J, Lorenz D H, Mezini M. Building modular object-oriented systems with reusable collaborations. In *Proc. International Conference on Software Engineering*, Limerick Ireland, June 4–11, 2000, pp.821–821.
[35] Allen E, Khoshgoftaar T. Measuring coupling and cohesion of software modules: An information theory approach. In *Proc. the Seventh International Software Metrics Symposium*, London, UK, April 4–6, 2001, pp.124–134.
[36] Mancoridis S, Mitchell B S, Chen Y, Gansner E R. Bunch: A clustering tool for the recovery and maintenance of software system structures. In *Proc. International Conference on Software Maintenance*, Oxford, England, IEEE Computer Society Press, 1999, pp.50–59.
[37] Misic V B. Cohesion is structural, coherence is functional: Different views, different measures. In *Proc. the Seventh International Software Metrics Symposium* (*METRICS-01*), London, UK, April 4–6, 2001, pp.135–144.
[38] Hautus E. Improving Java software through package structure analysis. In *Proc. International Conference on Software Engineering and Applications*, Cambridge, USA, Nov. 4–6, 2002.
[39] Lee J K, Seung S J, Kim S D, Hyun W, Han D H. Component identification method with coupling and cohesion. In *Proc. the Eighth Asia-Pacific on Software Engineering Conference* (*APSEC'01*), Macao, China, Dec. 4–7, 2001, pp.79–79.
[40] Abreu F B, Pereira G, Sousa P. A coupling-guided cluster analysis approach to reengineer the modularity of object-oriented systems. In *Proc. the 4th European Conference on Software Maintenance and Reengineering* (*CSMR'2000*), Zurich, Switzerland, Feb. 29–March 3, 2000, p.13.
[41] Abreu F B, Goulão M. A merit factor driven approach to the modularization of object-oriented systems. *L'Objet*, 2001, 7(4): 1–23.
[42] Szyperski C. Component Software: Beyond Object-Oriented Programming. New York: ACM Press/Addison-Wesley, 1998.
[43] Yu E, Mylopoulos J. An actor dependency model of organizational work — With application to business process reengineering. In *Proc. Conference on Organizational Computing Systems*, Milpitas, Calif., USA, Nov. 1–4, 1993, pp.258–268.

[44] Yu E. Towards modelling and reasoning support for early-phase requirements engineering. In *Proc. the 3rd IEEE Int. Symposium on Requirements Engineering* (*RE'97*), Washington DC, USA, Jan. 6–8, 1997, pp.226–235.

[45] Northcott M, Vigder M. Managing dependencies between software products. *Lecture Notes in Computer Science* 3412, Franch X, Port D (eds.), ICCBSS 2005, Springer-Verlag, Berlin/Heidelberg, 2005, pp.201–211.

[46] Grunske L, Kaiser B. An automated dependability analysis method for COTS-based systems. *Lecture Notes in Computer Science* 3412, Franch X, Port D (eds.), ICCBSS 2005, Springer-Verlag, Berlin/Heidelberg, 2005, pp.178–190.

[47] Franch X, Maiden N A M. Modeling component dependencies to inform their selection. *Lecture Notes in Computer Science 2580*, Erdogmus H, Weng T (eds.), ICCBSS 2003, Springer-Verlag, Berlin/Heidelberg, 2003, pp.81–91.

[48] Aggarwal K K, Singh Y, Chhabra J K. Complete dependency matrix for object-oriented software. *International Journal of Management and Systems* (*IJOMAS*), 2003, 19(1): 43–54.

[49] Gosling J, Yellin F. The Java Application Programming Interface. Vol. #1 (Core Packages)/#2 (Window Toolkit and Applets). Reading: Addison-Wesley, Massachusetts, USA, 1996.

[50] Briand L C, Daly J W, Wüst J K. A unified framework for coupling measurement in object-oriented systems. *IEEE Transactions on Software Engineering*, 1999, 25(1): 91–121.

[51] http://sourceforge.net/projects/xgen/.

[52] http://jakarta.apache.org.

[53] Kothari C R. Research Methodology: Methods & Techniques. Revised Second Edition, New Age International Publishers, New Delhi, 2007, pp.302–306.

Varun Gupta is pursuing his Ph.D. degree in the area of software engineering in Department of Computer Engineering, National Institute of Technology (Deemed University), Kurukshetra-136119 (INDIA). He obtained his Bachelor degree of Technology in computer science & engineering from Guru Nanak Dev University, Amritsar in 1999 and Master degree of Engineering in software engineering from Thapar Institute of Engineering and Technology, Patiala (Deemed University) in 2003. He worked as a lecturer in Department of Computer Science & Engineering, RIMT Institute of Engineering and Technology for 4 years. Presently, he is working as assistant director in Directorate of Information Technology, Punjab State Electricity Board, Patiala. His areas of interest include software engineering, object oriented design & development, and data mining.

Jitender Kumar Chhabra, Ph.D., is working as assistant professor in Department of Computer Engineering, National Institute of Technology (Deemed University), Kurukshetra-136119 (INDIA). He received his B.Tech. degree in computer engineering as 2nd rank holder and M.Tech. degree in computer engineering as Gold Medalist, both from Regional Engineering College, (now N. I. T.) Kurukshetra. He completed his Ph.D. degree in software metrics in GGS Indraprastha University, Delhi (INDIA). He has been teaching in N.I.T. Kurukshetra since last 15 years. He has also worked in collaboration with international software companies like Hewellet-Packard & Tata Consultacy Services. He has published more than 50 research papers in various international and national journals and conferences including IEEE, Elsevier, ACM. He is reviewer of many reputed research journals like IEEE and Elsevier etc. He has authored a widely circulated Schaum-Series book on Programming with C along with Byron S Gottfried from McGraw Hill Publications. He has been awarded with many prizes and awards including International Educator of year 2005, 2008, All India Open Debate Winner, Best Teacher Award, etc. His areas of interest include software engineering, database system, data structure, procedural and object oriented programming.