Journal of Theoretical and Applied Information Technology
20th November 2014. Vol. 69 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

DETERMINING SOFTWARE MAINTAINABILITY OF JAVA INTERFACES USING QUANTITATIVE APPROACH

[1]N.BASKAR, [2]A.V.RAMANI

[1]Research Scholar in Computer Science, SRMV College of Arts and Science, Coimbatore, India.
[2]Associate Professor, Department of Computer Science, SRMV College of Arts and Science, Coimbatore, India.
E-mail: [1]n_bas@rediffmail.com, [2]avvramani@yahoo.com

ABSTRACT

Software maintainability is one important aspect in the evolution of a software product. Several coupling measures have been introduced to identify and measure the design complexity of an object oriented (OO) systems. The coupling metrics proposed in this paper recognizes the complexity between inheritance and interface programming. This paper presents measurements of Coupling between Object (CBO) in object oriented programming. The metric values of class and interface inheritance diagrams have been compared to prove whether maintainability is improved to use and beneficial for the software developers.

Keywords: *Software Maintainability, Software Metrics, Object Oriented Systems, Interface, Coupling, Java*

1. INTRODUCTION

Maintainability is defined as the ease with which changes can be made to the software system. These changes are required for the correction of faults, adaptation of the system to a meet a new requirement, addition of new functionality, removal of existing functionality or corrected when errors or deficiencies occur and can be perfected, adapted or action taken to reduce further maintenance costs. Maintainability includes the notion of flexibility portability and transferability (from one development team to another). Maintainability of measure and monitor is an important task for mission-critical applications where changes can be made based on the tight time-to-market schedules and it is also important for IT to remain responsive business-driven changes to overcome the future needs [1]. It is also essential to keep maintenance costs under control. Software quality measurement is about quantifying to what extent software or system acquiring its desired characteristics. This can be evaluated through qualitative or quantitative approach or combining both approaches. In both cases, for each desirable characteristic, there are a set of measurable attributes the existence of which in a piece of software or system tends to be interrelated and associated to this characteristic. For example, an attribute associated with portability is the number of target-dependent statements in a program. The Software quality can be determined accurately using the Quality Function Deployment approach.

Software metric is defined as "a quantitative measure of the degree to which a system, component, or process possesses a given attribute."Software metrics can be categorized into direct measure and indirect measure. Direct measure of the software engineering process includes cost and effort applied. Direct measures of the product include lines of code (LOC) produced, execution speed, memory size, and defects reported over some set period of time. Indirect measure of the product includes functionality, quality, complexity, efficiency, reliability, maintainability [24]. Thus software

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

maintainability comes under indirect software measure.

An interface is a named collection of method definitions (without implementations). Interfaces are prototype for a class. Interfaces can be used like classes in declarations and signatures. An interface can also include constant declarations. Object references in Java may be specified to be of an interface type. They must either be null, or be bound to an object that implements the interface. 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 and integrate new features. Interfaces in OO programming contain names and signatures of methods and attributes, but it does not contain method implementations. Interfaces are used to organize code and provide a solid boundary between the different levels of abstraction.

The concept of interfaces has been measured in java programming by Fried Stiemann and Co [2] who denotes the usage of interfaces compared to classes are in the ratio of 4:1.Ken Pugh [3] states that obtaining commonality among classes makes it more effective for object oriented programming .He also explores the commonality in using inheritance and using interfaces in object oriented programming.

In this paper, the usage of interfaces is increased and the benefits of using interfaces are shown by coupling measures.

2. ESTABLISHING THRESHOLD FOR SOFTWARE MAINTAINABILITY

Software evolution has its own path other than that happening in natural biological world. The differences lie in the influential factors and how the factors interact in their evolutions. The software evolution is indivisible from software maintainability. The significant factors in software evolution can be computed as being equivalent to those of software maintainability so that the results of factor interaction are denotable by the probabilities of occurrence of maintenance behaviors. So according to ISO/IEC14764:2006, 2006 [4], software maintainability is all about change management and categorizes maintenance as following,

- Corrective Maintenance: If there is any fault after delivering the software projects they used to rectify the existing faults.
- Adaptive Maintenance: They used to adapt it to a changed or changing environment after the delivery of the software project;
- Perfective Maintenance: To improve performance or maintainability Any change to a software project after being delivered;
- Preventive Maintenance: Any change to a software project after being delivered to detect and correct any potential fault.

Therefore, it is reasonable to reckon different types of maintenance as the key factors influencing software maintainability and thereby software evolution.

3. OBJECT ORIENTED PROGRAMMING AND METRICS

Object oriented software is a more recent and important quality software than that of the old-style procedural software/program [7]. With the wide spread object oriented technology the subject of software engineering has received much attention over the last two decades [8] [9]. Object oriented design and development are very important and popular concepts in today's development environment. Object oriented design and development requires a different approach to design implementation and to the software metrics compared to standard set of metrics.

Object oriented metrics can be applied to analyze source code as an indicator of quality attributes. The source code may be any OO language. Metrics are very essential and important to measure object oriented software programming [10]. The development of software metrics for object oriented technology / programming has received more attention. A

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

large number of metrics have been developed by researchers and numerous tools are available to help assess design quality and to collect metrics from software programs, designs, quality and maintenance etc [11][12][13][7]. Many object oriented metrics proposed in literature survey lack theoretical proof and some have not been validated. These are the general metrics that are evaluating the object oriented programming concepts are: methods, classes, coupling and cohesion.

Degree of coupling between objects is a proposed OO metric. A higher degree of coupling between objects complicates application maintenance because object interconnections and interactions are more complex. The higher the degree of uncoupled object, the more objects can be reused within the same applications and also within other applications. Uncoupled objects should be easier to augment due to the lower degree of interaction. Testability is likely to degrade with a more highly coupled system of objects. Object interaction complexity associated with coupling can lead to increased error generation during development.

Average number of uses dependencies per object = total number of arcs / total number of objects.
A network is a collection of points, called vertices vertices, and a collection of lines, called arcs.
arcs = max (number of uses arcs) - in an object uses network
arcs - attached to any single object in a uses network.

Chidamber and Kemerer's [23] metrics suite for OO Design is the deepest research in OO metrics investigation. They have defined six metrics for the OO design. That the author described the approach of coupling between object classes (CBO) metric is used.CBO is defined as the count of the classes to which this class is coupled. Coupling is defined as two classes are coupled when methods declared in one class uses methods or instance variables of the other class. [Chidamber and Kemerer 1994]

Excessive coupling between object classes is harmful to modular design and prevents reuse. The more independent a class is easy to reuse it in another application. In order to improve modularity and promote encapsulation, inter-object class couples should be kept to a minimum. The larger the number of couples, the higher the sensitivity to changes in other parts of the design, and therefore maintenance is more difficult. A measure of coupling determines the testing of various parts of a design is complex. Very few metrics are presented for object oriented interfaces. In this paper, a measurement has been proposed to calculate the reusability of interfaces in object oriented programming.

4. SOFTWARE MAINTAINABILITY PREDICTION FOR JAVA INTERFACES

A model for software product quality has been formulated by associating a set of quality-carrying properties with each of the structural forms that are used to define the statements and components of a programming language. Software quality models are tools that lead to reliability enhancement activities to high risk modules for maximum effectiveness and efficiency. A software quality model predicts a quality factor such as the number of faults in a module, time for effective action. Software product and process metrics forms the basis of fault predictions. A software quality model includes the measurement of the properties of stability, analyzability, changeability and testability as sub-characteristics of a software product. Each sub-characteristic can be measured properly by many methods of metrics and each method of metrics can be applied to more than one sub-characteristic. By Multiplication Rule of Statistics, the indexes of all properties can be multiplied to produce a joint statistics of all the properties combined. Software product is the health status of a completed software product at the time of delivery.

Today Component Based Software Development (CBSD) is getting accepted in industry as a new effective development paradigm. It emphasizes the design and

Journal of Theoretical and Applied Information Technology
20th November 2014. Vol. 69 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

construction of software system using reusable components. CBSD is capable of reducing development costs and improving the reliability of an entire software system using components. The major advantages of CBSD are in-time and high quality solutions. Higher productivity, flexibility and quality through reusability, efficient maintainability, and scalability are some additional benefits of CBSD. It is also used for measuring the interfaces of the JAVA.

Software measurement plays an important role in finding the quality and reliability of software products. The measurement activities require appropriate tools to calculate relevant metric values. At present large number of metric tools are available for software measurement [5]. The main objective of this paper is to find the reusability of interfaces in object oriented programming.

4.1 Measurement and Metrics

Measurement is the technology that allows the software professional to make visible progress in improving the software related factors. Measurement is not only a performance factor that leads to behavioral changes but it is also used to improve the factors that are being measured [6]. Measurement is necessary for the software development process to be successful.

4.2 Object Oriented Interfaces

The concept of an interface is old. Software engineering has been using interfaces for more than 25 years. Nowadays interfaces are heavily used in all disciplines especially in object oriented programming [14]. Object oriented programming features become a good concept with high potential code reusability in the field of interface construct. Interfaces are used to organize code and provide a solid boundary between the different levels of abstraction [15] [16]. It is good to use interfaces in large type of applications because interfaces make the software/program easier to extend, 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 oriented programming 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 [17] [18] [19].

- Defining an Interfaces

Syntax

```
public interface InterfaceName
{
   // method signatures
}
```

Example:

```
public interface Measurable
{
   double getMeasure();
}
```

Purpose:

To define an interface and its method signatures.

The methods are automatically public

- Implementing the Interfaces

```
public class ClassName
   implements InterfaceName1,
   InterfaceName2, ...
{
// methods
      // instance variables
}
```

Example:

```
public class BankAccount
implements Measurable
{
   // Other BankAccount methods
   public double getMeasure()
   {
      // Method implementation
   }
}
```

Purpose:

To define a new class that implements the methods of an interface

4.3 Reusability

Reusability is always an interesting topic with shining promise. Reusable code is an effective combination 2 concept.

- Properly defined interface definitions and
- Efficiently defined class structure and inheritance.

Journal of Theoretical and Applied Information Technology
20th November 2014. Vol. 69 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

In this paper, the authors followed the first concept of reusability and measured the metric for interface reusability by giving a new formula. One benefit of defining interface is that every class that implements an interface must be inline with the interface's functional requirements. Large amount of code sharing occurs within each implementation classes. Based on the class structure designed at the development time the implementation classes are organized according to their interface group type and inheritance allowed to access common logic.

Reusability is an important factor for the software community people because it is the ability to reuse a number of software artifacts in terms of requirements, architecture, plans, cost estimates, designs, source code, data elements, interfaces, screens, user manuals, test plans and test cases. Software reusability is an experimental one under the impact of new tools and programming languages. The measurement of software/program and the software development process are much needed for software professionals attempting to improve their software process. Reusability of software increase productivity and quality and reduce the cost [6][20][21]. So in this paper, the reusability is measured for object oriented programming interfaces using the new formula.

In literature, relatively little information has been published on metrics. Those metrics would provide limited insight into the quality and usability of the interface [22].

So the proposed approach is to derive a formula for calculating the reusability of interfaces accurately. Deeper an interface in hierarchy leads to greater the reusability of inherited methods. When the depth of inheritance (DIT) of an interface increases the reusability of an interface also increases. So DIT of an interface has positive impact with the reusability of an interface. Reusability of interfaces are calculated by the following two ways:

(i) Reusability of interfaces is calculated by using the formula:

(RI)= Total Number of links to interfaces – Number of interfaces……..……. (1)

Where RI-Total Reusability of interface diagram.

(ii) The reusability of interfaces in a diagram is calculated by using the formula:-

Total Reusability of a diagram:

$RI=RI_1+RI_2+……+RI_n$……..…….(2)

Where R- Reusability and I_1.....I_n are Interfaces

In each diagram the reusability of an interface is calculated by using the formula and all interface reusability must be added to find the total reusability of interface diagram. By this way the values are calculated. First, it is not always the case that the reuse promoted by a given composition mechanism is going to lead to better maintainability. While the use of a specific mechanism can somehow contribute to modules reuse, it might also require developers to make various undesirable changes in their implementation. Second and more importantly, there are no metrics that enable to quantify the complexity properties of composition code. As a result, it is not possible to objectively assess to extent the intrinsic characteristics of composition mechanisms exert positive or negative influences on software reuse and maintenance. The class/metrics for reusability are

CBO: Coupling between object
NOC: Number of children's.
NASSocC: Number of associations between classes.
NDepIN: Number of dependencies in.
NDepOut: Number of dependencies out.
DIT: Depth of Inheritance Tree.

5. RESULTS

The results are compared between inheritance and interface coupling measures. The metrics discussed above are applied for both class inheritances and interface inheritance UML diagrams. The figure 1 is a vehicle classification class inheritance diagram. The figure 2 is a vehicle classification interface diagram.

Journal of Theoretical and Applied Information Technology
20th November 2014. Vol. 69 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

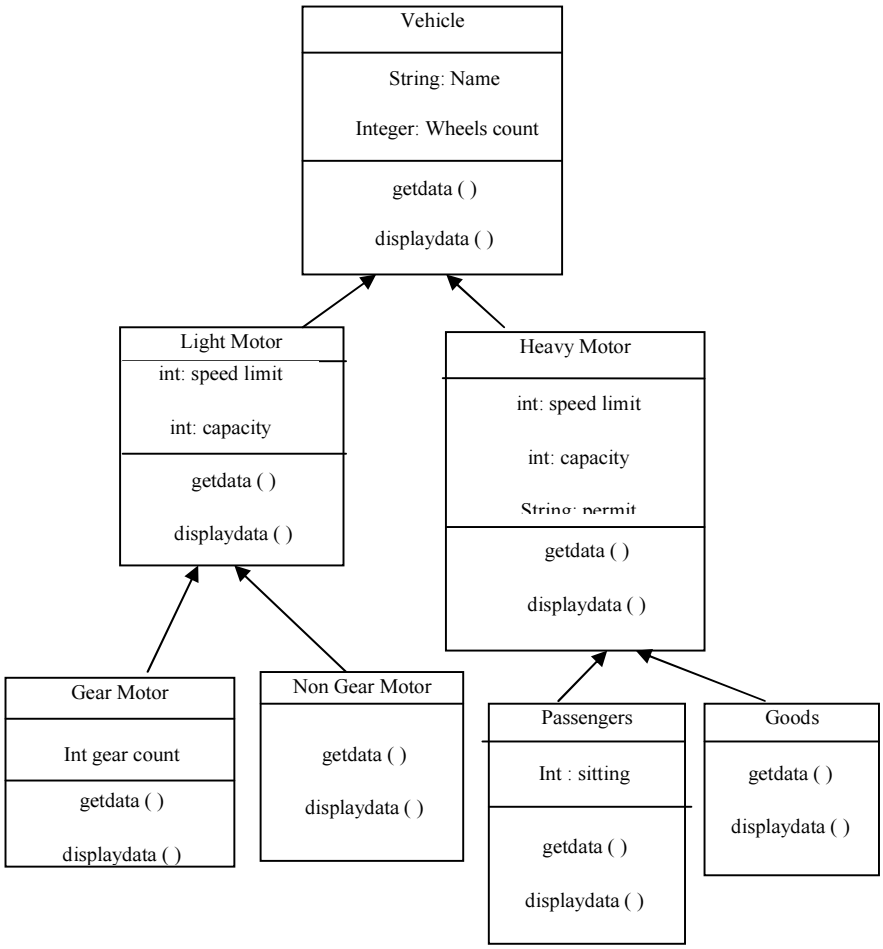
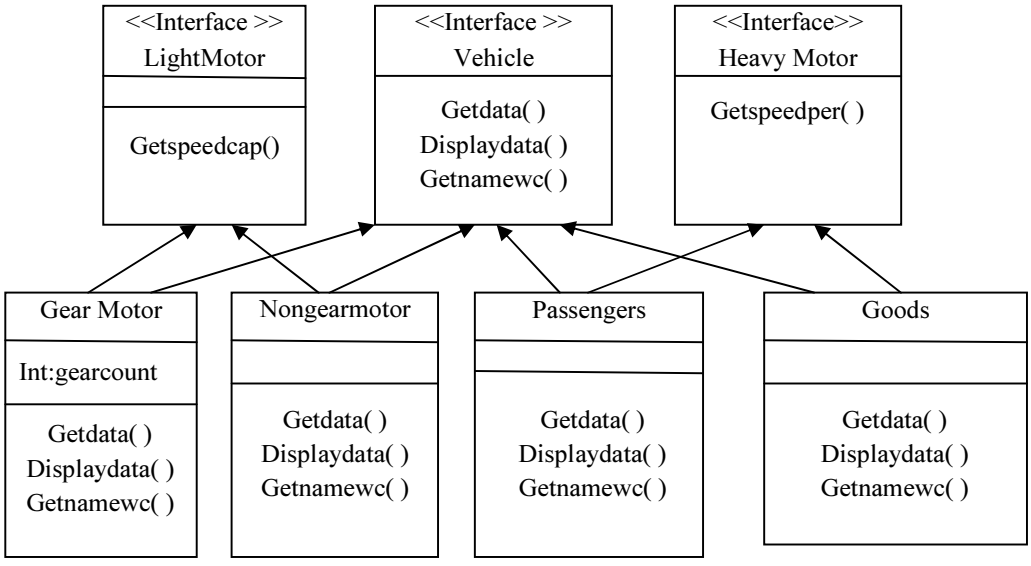


Figure 1: Vehicle Classification using Class Inheritance

The above said class inheritance Figure 1 is introduced with possible number of interfaces and is represented in Figure 2.



ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Figure 2: Vehicle Classification using Interfaces

For the above said two diagrams the coupling metrics are measured and tabulated in Table 1. By comparing the table values for both the diagrams the interface values are reduced for almost all metrics. And it is more efficient and accurate one.NOC is number of children in interface.

Table 1: Coupling Measures for Figure 1 & 2

	Metrics/ Classes	CBO	NAS SocC	NDe pIN	NDep Out	NOC	DIT
Class	Vehicle	2	2	2	0	2	0
	Light Motor	3	3	2	1	2	1
	Heavy Motor	3	3	2	1	2	1
	Gear Motor	1	1	0	1	0	2
	Non gear motor	1	1	0	1	0	2
	Passeng er	1	1	0	1	0	2
	Goods	1	1	0	1	0	2
Interface	Vehicle	1	1	1	0	1	0
	Light Motor	2	2	1	0	1	0
	Heavy Motor	2	2	1	0	0	0
	Gear Motor	1	1	0	1	0	1
	Non gear motor	1	1	0	0	0	1
	Passeng er	0	0	0	0	0	0
	Goods	0	0	0	0	0	0

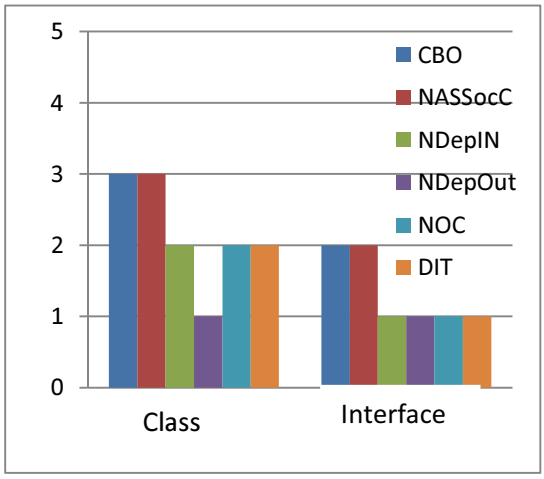


Figure 3: Coupling Measures for Figure 1 & 2

Figure 3 shows interface values are reduced for almost all metrics.

6. CONCLUSION

Proposed method reduces coupling in object oriented programming. Developers develop high quality program with the help of coupling. The benefits of this proposed approach are the increased flexibility and reduction of development cost. The most important in the software development is the reusability. Efficient metrics are used for the comparison of inheritance and interface concepts in object oriented programming. Interface concepts are better than class for increased usability and for maintenance. Object oriented programming software is more reusable than functionally decomposed software. As software is being developed, it is very important to keep an eye on the various parameters. The two UML object oriented diagrams are used to validate the formula. Hence, this approach is an eye-opener to measure reusability of interface diagram.

REFERENCES

[1] Boehm B. , "Software Engineering Economics", Prentice Hall (1981).
[2] Fried Stiemann, Wolf Siberski and Thomas Kuhne, " Towards the Systematic Use of Interfaces in Java Programming", 2nd Int.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Conf. on the Principles and practice of Programming in Java PPJ 2003, P.No:13-17.
[3] Ken Pugh," Interface Oriented Design", Chapter 5, 2005.
[4] Cem Kaner, and Walter P. Bond, "Software Engineering Metrics: What Do They Measure and How Do We Know?", 10TH INTERNATIONAL SOFTWARE METRICS SYMPOSIUM, METRICS, 2004.
[5] Bakar N.S.A.A. & Boughton .C,"Using a Combination of Measurement Tools to Extract Metrics from Open Source Projects", Proceedings of Software Engineering and Applications of 2008.
[6] Capers Jones," Applied Software Measurement-Global Analysis and Productivity Quality", 3rd Edition.
[7] 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.
[8] 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,March 29,2008.
[9] Terry .C. and Dikel .D.,"Reuse Library Standards Aid Users in Setting up Organizational Reuse Programs",Embedded System Programming Product News,1996.
[10] Linda H. Rosenberg,"Applying and InterpretingObject Oriented Metrics", Presented at the Software Technology Conference, Utah, April 1998.
[11] El Hachemi Alikacem, Houari A. Sahraoui, "Generic Metric Extraction Framework",IWSM/Metrickon, Software Measurement Conference 2006.
[12] NevilleI. Churcher, Martin J. Sheppered, ACM Software Engineering Notes, Vol.20, Issue 2, P.No:69-75, April 1995.
[13] Rudiger Lincke, Jonas Lundberg and Welf Lowe,"Comparing Software Metrics tools",ISSTA '08,July 20-24,2008.
[14] FriedRich Steimann, Philip Mayer, Andreas MeiBner,"Decoupling Classes with Inferred Interfaces ", Proceedings of 2006 ACM, Symposium on Applied Computing, Pg.No:1404-1408.
[15] Matthew Cochran,"Coding Better: Using Classes Vs. Interfaces", January 18th, 2009.
[16] Dirk Riehle and Erica Dubach,"Working With Java Interfaces and Markus Mohenen, "Interfaces with Default Implementations in Java", AachenUniversity of Technology.
[17] ISRD GROUP,"Introduction to Object Oriented Programming through JAVA",TATA Mc Graw Hill, Pg.No:109.
[18] Markus Mohnen,"Interfaces with Default Implementations in Java",Technical Report, RWTH Aachen,April 2002.
[19] Christopher L. Brooks, Christopher G.Buell, "A Tool for Automatically Gathering Object-Oriented Metrics", IEEE, 1994.
[20] Etzkorn W.E., Hughes, Jr W.E. and Davis C.G. ,"Automated reusability quality analysis of OO legacy software", Information and Software Technology, Volume 43 , Issue 5, April 2001,P.No:295-308.
[21] Khan R.A., K.Mustafa and S.A.Ahson, "Software Quality - Concepts And Practices", P.No:140.
[22] http://yunus.hun.edu.tr/~sencer/oom.html
[23] Classes-How to Separate Interfaces from Implementations", P.No:35-46, Published in Java Report 4, 1999.
[24] Roger S.Pressman,"Software Engineering a Practitioner's Approach", 6th Edition.