ITHEA

International Journal

INFORMATION THEORIES & APPLICATIONS

2009 Volume 16 Number 2

International Journal
INFORMATION THEORIES & APPLICATIONS
Volume 16 / 2009, Number 2

IJ ITA is official publisher of the scientific papers of the members of the ITHEA® International Scientific Society

IJ ITA welcomes scientific papers connected with any information theory or its application.

IJ ITA rules for preparing the manuscripts are compulsory.
The rules for the papers for IJ ITA as well as the subscription fees are given on *www.ithea.org* .
The camera-ready copy of the paper should be received by http://ij.ithea.org.
Responsibility for papers published in IJ ITA belongs to authors.

General Sponsor of IJ ITA is the Consortium FOI Bulgaria (www.foibg.com).

International Journal "INFORMATION THEORIES & APPLICATIONS" Vol.16, Number 2, 2009

Printed in Bulgaria

Edited by the Institute of Information Theories and Applications FOI ITHEA®, Bulgaria,
in collaboration with the V.M.Glushkov Institute of Cybernetics of NAS, Ukraine,
and the Institute of Mathematics and Informatics, BAS, Bulgaria.

Publisher: ITHEA®
Sofia, 1000, P.O.B. 775, Bulgaria. www.ithea.org, e-mail: info@foibg.com

ISSN 1310-0513 (printed) ISSN 1313-0463 (online) ISSN 1313-0498 (CD/DVD)

OBJECT LEVEL RUN-TIME COHESION MEASUREMENT

Varun Gupta, Jitender Kumar Chhabra

Abstract: Most of the object-oriented cohesion metrics proposed in the literature are static in nature and are defined at the class level. In this paper, new dynamic cohesion metrics are proposed which provide scope of cohesion measurement up to the object level and take into account important and widely used object-oriented features such as inheritance, polymorphism and dynamic binding during measurement. The proposed dynamic measures are computed at run-time, which take into consideration the actual interactions taking place among members of a class. The proposed measures are evaluated using a theoretical framework to prove their usefulness. A dynamic analyzer tool is presented which can be used to perform dynamic analysis of Java applications for the purpose of collecting run-time data for the computation of the proposed metrics. Further, a case study is conducted using a Java program to demonstrate the computation process for the proposed dynamic cohesion measures.

Keywords: Cohesion; Software metrics; Static metrics; Dynamic metrics; Aspect oriented programming; Object-oriented software.

ACM Classification Keywords: D.2.8 [Software Engineering]: Metrics; D.2.3 [Software Engineering]: Coding Tools and Techniques - Object-oriented programming

Introduction

High quality software, among many other principles, should obey the principle of high cohesion. Stevens et al. [1], who first introduced cohesion in the context of structured development techniques, define cohesion as a measure of the degree to which the elements of a module belong together. In a highly cohesive module, all elements are related to each other for performing a single function. The higher the cohesion of a module, the easier the module is to develop, maintain, and reuse, and the less fault-prone it is [2], [3], [4]. The principle of high cohesion has been transferred to object-oriented software by Coad and Yourdon [5, 6] and research in this field has lead to a large number of cohesion measures for object-oriented systems being defined [7-21]. However, most of the cohesion metrics proposed in the literature for measurement of cohesion are static in nature and static cohesion metrics may be insufficient in evaluating the dynamic behavior of an application at runtime, as its behavior will be influenced by the execution environment as well as the complexity of the source code. Object-oriented features such as polymorphism, dynamic binding, inheritance and common presence of unused code in commercial software, cause the static metrics to be inaccurate, as they do not precisely reflect the run-time situation of the software [22]. Moreover, the complex dynamic behavior of many real-time applications motivates us to focus on dynamic cohesion metrics in place of static cohesion metrics. Dynamic cohesion metrics are obtained from the execution traces of the code or from the executable models. Till date, only a few dynamic metrics have been proposed for the measurement of cohesion. Gupta et al. [23] have proposed program execution based module cohesion metrics based on the dynamic slicing of the program. However, these metrics only deal with procedure-oriented program. The run-time cohesion metrics proposed by Mitchell et al. [24, 25] are just dynamic equivalent of the existing cohesion metrics such as LCOM given by Chidamber and Kemerer [7].

The remainder of the paper is organized as follows. Section 2 discusses advantages of dynamic cohesion measures over static cohesion measures and Section 3 contains the definitions of the proposed dynamic

cohesion measures. In Section 4, the proposed measures are validated theoretically and Section 5 provides a dynamic analyzer tool for computation of the proposed measures. In Section 6, a case study is conducted to demonstrate the process of computation of the proposed measures and finally, Section 7 concludes the work.

Advantages of Dynamic Cohesion Metrics

Static cohesion metrics are obviously simpler to collect because there is no need to run the software. Moreover, to obtain dynamic cohesion metrics, code or simulation model of the software system is needed, which is available very late in the development life cycle. Static cohesion metrics are widely used due to the fact that they are easier to obtain, especially at the early stages of software development. But, the potential benefits of dynamic cohesion metrics collected by executing the program outweigh the complexity and cost of measuring them. The ability to static cohesion metrics to measure the quality attributes of a software system is less apparent, as the static cohesion metrics are evaluated only by means of static inspection of the software artifact. Since, it is the actual runtime behavior of the system that determines its quality, not the potential characteristics implied by the static analysis of the software system and dynamic cohesion metrics are computed based on the data collected during actual execution of the system, and thus directly reflect the quality attributes (performance, change-proneness, error rates etc.) of the software in its operational mode. Moreover, static cohesion metrics deal with the structural aspects of a software system, whereas dynamic cohesion metrics also deal with the behavioral aspects of the system. Moreover, static cohesion metrics are somewhat constrained in their ability to deal with inheritance, polymorphism and dynamic binding issues since the run-time types at field access and method invocation sites are not known, whereas dynamic cohesion metrics are capable to deal with such issues.

Dynamic Cohesion Metrics

The mapping level of dynamic cohesion measurement can be either object or class. Object-level dynamic cohesion quantifies the extent of dependencies between the members of an object at run-time. As object is an instance of a class created at runtime, class-level dynamic cohesion aggregates the object-level cohesion values of all instances of a class. As an object or a class consists of two types of elements i.e. attributes and methods and there are mainly two kinds of dependencies among elements of an object or a class: (i) dependency between attributes and methods, and (ii) dependency between a pair of methods. First types of dependency exists due to read and write types of interactions present between methods and attributes i.e. when a method reads or writes the value of an attribute. Second type of dependency takes place due to the presence of call type of interactions between methods i.e. when a method calls other method of the class or object. However, not all the methods of a class contribute to its cohesion [21]. There exist some special methods such as constructor, destructor, access methods and delegation methods intrinsically accessing only some of the attributes in the class [17], [18], [21]. A constructor is a type of method that initializes essential attributes of the class and a destructor is a type of method that may only de-initialize crucial attributes of the class. An access method is a method that only reads or writes a particular attribute of the class. A delegation method is a method that only delegates a message to another object, especially to an attribute in the class, thus, generally have only one interaction with one attribute. These special methods may not essentially access all of the attributes. It has been widely accepted by a number of authors that these methods have no influence on the cohesion of a class [4], [17], [18], [21]. Thus, these methods need to be excluded in the measurement of cohesion of a class. The cohesion measurement of an object or class should consider only two types of elements: normal methods (except special methods) and attributes and two types of dependencies between elements i.e. access relations between normal methods and attributes and call relations between pairs of normal methods. Thus, the dynamic cohesion of an object or class should be measured from the two aspects.

Dynamic Access Cohesion

Dynamic access cohesion exists between methods and attributes of an object when a method of an object reads or writes an attribute of the same object during execution of the program. This type of dynamic cohesion for an object o is defined as the ratio of actual number of distinct dependence relations between all methods and all attributes to the maximum possible number of dependence relations of this type between them (i.e. $n \times m$). In case, if either number of methods or number of attributes are zero for an object then this type of cohesion would be nil for that object. This kind of dynamic cohesion for an object is defined as follows: -

$$DC_{ACC}(o) = \begin{cases} 0 & \text{n=0 or m=0} \\ \dfrac{\sum_{i=1}^{n}\sum_{j=1}^{m} Dep_R(m_i, a_j)}{n \times m} & \text{n} \neq 0 \text{ and m} \neq 0 \end{cases}$$

Where $Dep_R(m_i, a_j)$ is the access dependency present at run-time between a method m_i and an attribute a_j of an object o. Also, n is the total number of methods of object o and m is the total number of attributes of object o.

Dynamic Call Cohesion

Dynamic call cohesion exists between a pair of methods of an object when a method m_i calls other method m_j of the object during program execution. This kind of dynamic cohesion of an object o is defined as the ratio of actual count of distinct dependence relations between all ordered pairs of methods to the maximum possible number of relations of this type between them (i.e. $n \times (n-1)$). In case, if number of methods of an object is zero then this type of cohesion is also zero for that object and if a single method exists for an object, then this type of cohesion is maximum i.e. 1 for that object. This form of dynamic cohesion for an object o is defined as follows: -

$$DC_{CALL}(o) = \begin{cases} 0 & \text{n} = 0 \\ \dfrac{\sum_{i=1}^{n}\sum_{j=1 \wedge j \neq i}^{n} Dep_R(m_i, m_j)}{n \times (n-1)} & \text{n} \neq 0 \\ 1 & \text{n} = 1 \end{cases}$$

Where $Dep_R(m_i, m_j)$ is the call dependency present at run-time between methods m_i and m_j of an object o. Also, n is the total number of methods of object o.

Weight-age of Different Types of Cohesion

There are two types of dynamic cohesion for an object as defined above. All these types of cohesion have got different weight-ages due to the different types of dependence relations attached with them. The weight-ages of these types of cohesion having defined after taking into consideration their relative ordering as well as expert developers' opinions and are given in Table 1. The weights to these types of cohesion are assigned as per the intensity of the relation attached with them. First, the dynamic cohesion due to access dependency between methods and attributes (DC_{ACC}) is more significant than dynamic cohesion due to call dependency between methods (DC_{CALL}) due to the fact that most of the cohesion measures are defined in terms of degree of

interactions among methods and attributes [7], [11], [8], [13], [9], and [10]. Thus, DC_{ACC} has got more weight-age than DC_{CALL}.

Table 1 Weight-age of different types of cohesion

Dynamic Cohesion Type	Weight-age
Dynamic access cohesion (DC_{ACC})	2
Dynamic call cohesion (DC_{CALL})	1

Object Level and Class Level Cohesion Measures

Object level dynamic cohesion for an object is defined as the weighted summation of two types of cohesions defined above. The Dynamic cohesion for an object o is defined as:

$$ODC(o) = \frac{w_1 * DC_{ACC}(o) + w_2 * DC_{CALL}(o)}{w_1 + w_2}$$

Where, w_1=2 and w_2=1

Class level Dynamic Cohesion for a class is defined as the average of the values of Object level Dynamic Cohesion for all objects of a class created at run-time i.e.

$$CDC(c) = \frac{\sum_{i=1}^{k} ODC(o_i)}{k}$$

Where *k* is the number objects of class created at run-time.

Theoretical Validation

The purpose of this section is to validate the proposed measures theoretically by using the four properties given by Briand et al. [26]. The four cohesion properties defined by Briand et al. characterize cohesion in a reasonably intuitive and rigorous manner. A well-defined cohesion measure should have the following four properties. These properties provide a guideline to develop a good cohesion measure.

Property 1 (Non-negativity and Normalization). Normalization of a cohesion measure makes it possible to carry out meaningful comparisons between the cohesion values of classes or objects having different number of elements, since they all belong to the same interval [6]. As per the definitions of the above-defined measures, the cohesion of an object or a class c lies within a specified range i.e. $ODC(o) \in [0,1]$ and $CDC(c) \in [0,1]$. Thus, Property 1 holds for the proposed cohesion measures.

Property 2 (Null value). This property states that if there is no dependency among the members of an object or class, then the cohesion of that object or class should be null. As per the definitions of the proposed measures, if there is no dependency relation between the elements of an object at run-time, then the values of the two types of cohesions for an object o i.e. DC_{ACC} (o) and DC_{CALL}(o) will certainly be zero and as a result dynamic cohesion of an object o, ODC(o) will also be zero as ODC(o) is the weighted summation of the above measures only. Moreover, the cohesion of a class c will also be null if cohesion values of all objects of the class are null. Thus, the proposed measures satisfy Property 2.

Property 3 (Monotonicity). This property requires that by addition of dependency relationships among elements of an object or a class should not decrease its cohesion.

Let object, $o = <E^R, R^R>$, where R^R represents the set of relations among set of elements, E^R of an object o at run-time. Let a relationship is added to o to form a new object $o' = <E^R, R^{R'}>$, which is identical to o except that $R^R \subset R^{R'}$. Then, as per the above given definitions of the measures, dynamic cohesion value of new object will only increase or will remain the same but will never decrease.

For objects, $o = <E^R, R^{R'}>$ and $o' = <E^R, R^{R'}>$, if $R^R \subset R^{R'}$ then $ODC(o) \leq ODC(o')$. Similarly, the property holds at class level also. Thus, the proposed measures satisfy this property as well.

Property 4 (Merging of objects or classes). This property states that the cohesion of an object or a class obtained by putting together two unrelated objects or classes is not greater than the maximum cohesion of the two original objects or classes. If two unrelated objects o_i and o_j are merged to form a new object o_k then the cohesion of o_k is no larger than the maximum cohesion of o_i and o_j or if two unrelated classes c_i and c_j are merged to form a new class c_k, then cohesion of c_k is no larger than the maximum cohesion of c_1 and c_2.

For, $o_i = <E_i^R, R_i^{R'}>$ and $o_j = <E_j^R, R_j^R>$ where $R_i^R \cap R_j^R = \phi$.

and $c_i = <E_i, R_i>$ and $c_j = <E_j, R_j>$ where $R_i \cap R_j = \phi$

Since, two unrelated objects or classes have been combined to form a new object or class; there is a proportionate increase in number of dependency relations as well as in number of elements. As per the definition of cohesion measures which measure cohesion in terms of ratio of actual number of dependence relations existing at run-time divided by the maximum possible number of relations among elements. There is no net increase in the value of cohesion measure since numerator values as well as denominator values have increased together. Thus, the cohesion value of the combined object or class cannot be more than the maximum of the two unrelated objects or classes i.e.

$Max\{ODC(o_i), ODC(o_j)\} \geq ODC(o_k)$ and $Max\{CDC(c_i), CDC(c_j)\} \geq CDC(c_k)$

Hence, the proposed cohesion measures satisfy property 4 also.

Dynamic Analyzer for the Proposed Measures

We used aspect-oriented programming (AOP) approach for dynamic analysis of object-oriented programs for the purpose of computation of the proposed measures, as AOP is an efficient technique for dynamic analysis without any side effects [27]. We used AspectJ [28] to develop a dynamic analyzer tool dynamic analysis of Java applications for computation of the proposed measures. We have written an aspect using Aspectj for dynamic analysis of target Java programs and this aspect is an independent programming unit and can be merged with the target Java programs without altering the behavior of the target programs. Figure 1 presents the key features of the dynamic analyzer tool implemented using AspectJ.

```
public aspect DynamicCohesionAnalyser{

//Pointcuts defined
pointcut traceMethods() : (execution (* *.*(..)) ) …
pointcut traceAttribs() : ( (get(* *) || set(* *) ) …
pointcut traceAccess() : ( (get(* *) || set(* *) )  && withincode(* *.*(..)) …
```

```
pointcut traceCall () : call(* *.*(..))  …

// Advices defined for capturing pointcuts
before(): traceMethods(){
Signature sig=thisJoinPointStaticPart.getSignature();
…
 }

after(): traceAttribs(){
Signature sig=thisJoinPointStaticPart.getSignature();
…
}
after(): traceAccess() {
…
}
   before(): traceCall (){
…
}

// methods storing data collected at run-time into files
void writeToFile_traceCall(Signature sig)
{
…
}

void writeToFile_traceCall (Signature sig)
{
…
}
} //aspect
```

Figure 1 Main features of the Dynamic Analyzer Tool

Case Study

In this section, a case study is carried out to demonstrate the process of computation of the proposed dynamic cohesion measures using a program written in Java [29] shown in Figure 2. This program consists of a class ArrayQueue [30]. This class consists of four attributes and seven normal methods.

```
public class ArrayQueue {
private Object [ ] theArray;
private int  currentSize;
private int front;
private int back;
 public ArrayQueue( )    {
 theArray = new Object[10];
        makeEmpty( );    }
 public boolean isEmpty( )    {
        return currentSize == 0;    }
public void makeEmpty( )    {
        currentSize = 0;
        front = 0;
        back = -1;    }
```

```
public Object dequeue( )    {
        if( isEmpty( ) )
            throw new UnderflowException( "ArrayQueue dequeue" );
        currentSize--;
       Object returnValue = theArray[ front ];
        front = increment( front );
        return returnValue;    }
   public Object getFront( )    {
        if( isEmpty( ) )
            return theArray[ front ];    }
 public void enqueue( Object x )    {
        if( currentSize == theArray.length )
            doubleQueue( );
        back = increment( back );
        theArray[ back ] = x;
        currentSize++;    }
 private int increment( int x )    {
        if( ++x == theArray.length )
            x = 0;
        return x;    }
 void doubleQueue( )    {
        Object [ ] newArray;
        newArray = new Object[ theArray.length * 2 ];
        for( int i = 0; i < currentSize; i++, front = increment( front ) )
            newArray[ i ] = theArray[ front ];
        theArray = newArray;
        front = 0;
        back = currentSize - 1;    }
public static void main(String str[]) {
ArrayQueue q1=new ArrayQueue(5);
…
…
}//main method
} //ArrayQueue Class
```

Figure 2 Java program [29]

On the execution of the above program along with the dynamic analyser tool, values of different cohesion measures obtained are as follows: -

DC_{ACC} (q1) =0.61

DC_{CALL} (q1) = 0.07

Thus, dynamic cohesion values for object q1 and class ArrayQueue are calculated as follows: -

ODC (q1) = (2*0.61+1*0.07)/3= 0.84

CDC (ArrayQueue) =0.84

Conclusion

This paper proposes new well-defined dynamic cohesion measures which satisfy the four cohesion properties defined by Briand et al. and in comparison with the existing cohesion measures, the proposed measures have the following advantages: -

- The proposed dynamic cohesion measures are more accurate as they are defined at run-time and take into consideration the actual interactions taking place rather than the potential interactions which may or may not happen as is the case with static cohesion metrics.
- The proposed cohesion metrics take inheritance and polymorphism into consideration as the actual targets of polymorphic invocations can only be determined at run-time due to the presence of inherited members of a class.
- The scope of measurement of the proposed dynamic cohesion metrics can be specific to a single object. Whereas, other existing cohesion metrics are able to measure cohesion up to the class level only.

In future work, we plan to conduct some empirical study and compare these new dynamic cohesion measures with the existing static and dynamic cohesion measures to prove that the proposed measures are better indicators of dynamic cohesion in comparison to the existing metrics.

Bibliography

[1] W. Stevens, G. Myers, and L. Constantine, Structured design, *IBM Systems J.* 13(2) (1974) 115–139.

[2] D.N. Card, V.E. Church, and W.W. Agresti, An empirical study of software design practices, IEEE Transactions on Software Engineering 12(2) (1986) 264-271.

[3] L. Briand, S. Morasca, and V. Basili, Defining and validating high-level design metrics, Technical Report, University of Maryland, CS-TR 3301, 1994.

[4] L.C. Briand, J.W. Daly,and J. Wust, A unified framework for cohesion measurement in object-oriented systems, Empirical Software Engineering 3(1) (1998) 65–117.

[5] P. Coad and E. Yourdon, Object-Oriented Analysis (Prentice Hall, 1991).

[6] P. Coad and E. Yourdon, Object-Oriented Design (Prentice Hall, 1991).

[7] S.R. Chidamber and C.F. Kemerer, A metrics suite for object-oriented design, IEEE Transactions on Software Engineering 20(6) (1994) 476–493.

[8] J. Bieman and B. Kang, Cohesion and reuse in an object-oriented system, In: Proceedings of the ACM Symp. Software Reusability (SSR'95), (1995) 259–262. Reprinted in ACM SIGSOFT Software Engineering Notes (1995).

[9] L.M. Ott, J.M. Bieman and B.K. Kang, Developing measures of class cohesion for object-oriented software, In: Proceedings of the 7th Annual Oregon Workshop on Software Metrics, (Oregon, Portland, 1995).

[10] L.M. Ott and J.M. Bieman, Program slices as an abstraction for cohesion measurement, Journal of Information and Software Technology 40(11–12) (1998) 691–699.

[11] M. Hitz and B. Montazeri, Measuring coupling and cohesion in object oriented systems, In: Proceedings of the International Symposium on Applied Corporate Computing, (Monterrey, Mexico, 1995) 25–27.

[12] Y.S. Lee and B.S. Liang, Measuring the coupling and cohesion of an object-oriented program based on information flow, In: Proceedings of the International Conference on Software Quality, (Maribor, Slovenia, 1995) 81–90.

[13] B. Henderson-Sellers, Software Metrics, (Prentice Hall, Hemel Hempstaed, UK, 1996).

[14] S. Moser, V.B. Misic, Measuring class coupling and cohesion: a formal meta- model approach, In: Proceedings of the Asia Pacific Software Engineering Conference and International Computer Science Conference, (IEEE Computer Society Press, Hong Kong, 1997) 31–40.

[15] S. Counsell, E. Mendes, S. Swift, A. Tucker, Evaluation of an object-oriented cohesion metric through Hamming distances. Tech. Rep. BBKCS-02-10, Birkbeck College, University of London, UK, 2002.

[16] J. Bansiya, L.H. Etzkorn, C.G. Davis, W. Li, A class cohesion metric for object oriented designs, Journal of Object-oriented Programming 11(8) (1999) 47–52.

[17] H.S. Chae and Y.R. Kwon, A cohesion measure for classes in object-oriented systems, In: Proceedings of the Fifth International Software Metric Symposium (METRICS'98), (Bethesda, MD, USA, IEEE Computer Society Press, 1998) 158–166.

[18] H.S. Chae, Y.R. Kwon, D.H. Bae, A cohesion measure for object oriented classes, *Software Practice and Experience* 30(12) (2000) 1405–1431.

[19] Z. Chen, Y. Zhou, B. Xu, J. Zhao, H. Yang, A novel approach to measuring class cohesion based on dependence analysis, In: *Proceedings of the International Conference on Software Maintenance*, (IEEE Computer Society Press, Montreal, Canada, 2002) 377–384.

[20] Y. Zhou, B. Xu, J. Zhao and H. Yang, ICBMC: an improved cohesion measure for classes, In: *Proceedings of the International Conference on Software Maintenance*, (IEEE Computer Society Press, Montreal, Canada, 2002) 44–53.

[21] J. Wang, Y. Zhou, L. Wen, Y. Chen, H. Lu and B. Xu, DMC: a more precise cohesion measure for classes, *Information and Software Technology* 47(3) (2005) 167-180.

[22] A. Mitchell, J.F. Power, An Empirical Investigation into the Dimensions of Run-Time Coupling in Java Programs, In: *Proceedings of the 3rd Conference on the Principles and Practice of Programming in Java*, (Las Vegas, Nevada, 2004) 9–14.

[23] N. Gupta and P. Rao, Program execution based module cohesion measurement, In: *Proceedings of the 16th International Conference on Automated on Software Engineering (ASE '01)*, (San Diego, USA, 2001).

[24] A. Mitchell and J.F. Power, Run-Time Cohesion Metrics for the Analysis of Java Programs, *Technical Report Series no. NUIM-CS-TR-2003-08, National University of Ireland*, (Maynooth, Co. Kildare, Ireland, 2003).

[25] A. Mitchell and J.F. Power, Run-Time Cohesion Metrics: An Empirical Investigation, *In: Proceedings of the SERP*, 2006.

[26] L.C. Briand, S. Morasca, and V.R. Basili, Property-based software engineering measurement, *IEEE Transactions on Software Engineering* 22(1) (1996) 68–85.

[27] V. Gupta and J.K. Chhabra, Measurement of dynamic metrics using dynamic analysis of programs, *In: Proceedings of the WSEAS International Conference on Applied Computing Conference*, Istanbul, Turkey, (2008) 81-86.

[28] AspectJ, Available at http://www.eclipse.org/aspectj (accessed 05 Dec 2009).

[29] P. Naughton and H. Schildt, Java 2: The Complete Reference, 3rd revised edition (McGraw-Hill, 1999).

[30] ArrayQueue implementation in Java, Available at http://www.java-tips.org (accessed 15 Dec 2009).

Authors' Information

Varun Gupta – Researcher; Department of Computer Engineering, National Institute of Technology, Kurukshetra, Kurukshetra-136119 India; e-mail: varun3dec@yahoo.com

Major Fields of Scientific Research: Software Engineering, Object Oriented Design & Development, Aspect Oriented Programming.

Jitender Kumar Chhabra – Astt. Professor; Department of Computer Engineering, National Institute of Technology, Kurukshetra, Kurukshetra-136119 India;

e-mail: jitenderchhabra@rediffmail.com

Major Fields of Scientific Research: Software Engineering, Database System, Data Structure, Procedural and Object-Oriented Programming.