



International Journal of Emerging Technologies in Computational and Applied Sciences (IJETCAS)

www.iasir.net

Dynamic cohesion metrics for aspect oriented programming

Himanshu Dua¹, Puneet Jai Kaur²
University Institute of Engineering and Technology
Panjab University, Chandigarh, India

Abstract: *Classes and Aspects are basic unit in AOP. These qualities influenced the readability, maintainability and also affect the degree of relatedness among classes and aspect members. High cohesion is desirable characteristic. Existing techniques for measuring cohesion do not demonstrates actual cohesion of all the AOP cohesion metric, none of them are dynamic in nature. In this paper, a new dynamic cohesion metrics which is extension of most widely used static metrics in OOP i.e. LCOM is presented. A dynamic evaluation tool is developed using AspectJ(JAVA) and Python to gather runtime data to estimate the dynamic cohesion. This new discussed cohesion metrics gives more appropriate vision to the behavioral aspects of the system.*

Keywords: *Aspect Oriented Programming (AOP); dynamic cohesion metrics; AspectJ; Lack of cohesion in operations(LCOO);object oriented programming(OOP).*

I. Introduction

As factors like reusability and maintainability are not easily met in large applications that's why new methodologies and tools are developed. Maintainability and reusability plays a big role in deciding the cost of software. Whenever a new module is added during software development process checking its maintainability and reusability should be a mandatory constraint. Managing and developing software often use several metrics to measure and enhance the quality metrics.

There are several software parameters by which software quality can be measured such as coupling, cohesion, polymorphism and complexity. Maintainability and reusability are the factors which are provided by cohesion [1] [2]. Higher cohesion is a desirable property which means it is difficult to split a class and almost all members of a class are related to each other. It has been proved that cohesion metrics plays significant role in assessment of design quality [3], fault prediction [4] [5] [6], modularization of software [7] [8] and productivity, design and reuse effect [9].

Most of the cohesion metrics which has been developed provide static values of cohesion. Only some of them use runtime data to evaluate cohesion value [10] [11] [12]. Out of these, most are direct extension of static metrics and a few like Kumar metrics are from scratch [13]. The dynamic metrics proposed in this paper is an extension of LCOM which was developed by C&K. main advantage of dynamic metrics is that it collects data at run time and then evaluates the metric value on runtime data rather than static data. For instance, there is a method which has a very low cohesion and if it is not called at run time or even if called just once this does not impact on the overall software quality of a system.

Scope of dynamic metrics varies than the scope of static cohesion metric. Scope of static metrics is specific to whole class whereas scope of dynamic cohesion metrics is also the object of that class in addition to the whole class. Static cohesion metrics try to anticipate the probable interactions that would take place at runtime whereas dynamic cohesion metrics measures what is really occurring at runtime rather than anticipating, dynamic cohesion metrics are likely to reflect the software design more accurately than static cohesion metric.

II. Related Work

There are number of cohesion metrics which have been proposed for OOP. Some metrics are static and some are dynamic. But only a few metrics have been proposed for AOP, which are all static metrics. There are three main metrics which have been proposed for AOP which are LCOO, dependency metrics and ACoH metrics.

LCOO is an extension of most widely used cohesion metric. LCOM metrics is used in OOP. LCOO measure the lack of cohesion in software. It is proposed by Sant Anna in 2003. LCOO uses instance variable to evaluate cohesion.

Lower the value of LCOO indicates higher cohesion and vice versa. It does not reflect the true cohesion of the system. The second main cohesion metric is dependency metric which is proposed by Zhao and Xu in 2004 [19]. It evaluates value among the attributes and modules. It considers three types of dependencies; inter attribute dependencies, inter-module dependencies and module –attribute dependencies. The dependencies are good measure of cohesion but it is very complex to evaluate these dependencies in large software's.

The other metrics which has been proposed for AOP is Acoh[14] which is inspired by class cohesion measurement [15]. It evaluates the cohesion value based on relatedness among the module. It considers two types of connection module-data connection and module-module connection. It considers both direct and indirect connection. It reflects the quality design of software.

These three metrics are static in nature. They evaluate the metric value of static data. These three metrics are inadequate to provide information about the true cohesion of software. There is some research which shows that static metrics are not capable of measuring the dynamic aspects of software [16] [17].

This work proposes a new dynamic metrics which evaluate the cohesion on runtime data. It evaluates the cohesion value based on relatedness among the module.

III. Proposed Work

The work has been done in five steps. The steps include running the AspectJ project, then analyzing the program at run time, followed by saving the sequential execution of functions and variables in a text file. After this the text file is read in order to search the variables which are called at runtime. Finally the value of LCOO is calculated to find the cohesion of the system. Here the project we have considered is the 'Aspect project for Telecom industry' that collects the information of the amount of time a user talks to another. Analysis of the program has been done by using pointcuts and joinpoints of AOP. The sequential execution has been saved in a text file. For searching the variables that are called at runtime Python file handling has been used.

In the end the value of LCOO has been calculated, based on the variables that are called at run time. The explanation of the above steps is as follows:

Step1: To study an AspectJ Project:

An example system comprises a simple model of telephone connections to which timing and billing features are added using aspects. In this the billing feature depends upon the timing feature. The example application is a simple simulation of a telephony system in which customers make, accept, merge and hang-up both local and long distance calls. The application architecture is in three layers. The basic objects provide fundamental functionality to simulate customers, calls and connections (regular calls have one connection, conference calls have more than one). The timing feature is concerned with timing the connections and keeping the total connection time per customer. Aspects are used to add a timer to each connection and to manage the total time per customer. The billing feature is concerned with charging customers for the calls they make. Aspects are used to calculate a charge per connection and, upon termination of a connection, to add the charge to the appropriate customer's bill.

Step2: Analyze the project at run time:

AspectJ is very useful to provide dynamic analysis of a program. A pointcut can be written to provide the functions and the variables called at run time.

```
pointcut myClass(): within(telecom.*);
```

This pointcut intersect the class at run time whenever any class in the telecom package calls any function or the variable.

Step 3: Saving the execution sequence of methods at run time.

Now we can easily save the execution sequence in a file with file handling in java. We will use FileWriter and BufferedWriter class of java to write output in file. FileWriter class inherits from the OutputStreamWriter class. The class is used for writing streams of characters. This class has several constructors to create required objects. The syntax to create a FileWriter object when a File object is given below:

```
FileWriter(File file)
```

Step 4 : Reading data from file:

For reading a file python has been used because it is a high level language and has very powerful python regular expression. Regular expressions are a powerful language for matching text patterns. The Python "re" module provides regular expression support.

In Python a regular expression search can be done as:

```
match = re.search(pattern, string)
```

The re.search() method takes a regular expression pattern and a string and searches for that pattern within the string. If the search is successful, search() returns a match object else it returns None. Therefore, the search is usually immediately followed by an if-statement to test if the search succeeded, as shown in the following example which searches for the pattern 'word:' followed by a 3 letter word (details below):

```
str = 'an example word:Test!'
```

```
found = re.search(r'word:\w\w\w', str)
```

```
if found:
```

```
    print 'found', found.group()
```

```
else:
```

```
    print 'did not find'
```

The code `found = re.search(pattern, string)` stores the search result in a variable named "found". Pattern given is 'word:\w\w\w\w ' and the string is 'an example word:Test!!'. Firstly python searches for 'word:' in the string then checks the next four positions for the any alphabets letter. The function `found.group()` return the 'word:Test'.

Step5: Calculate the value of LCOO:

One can easily find the value of LCOO if one knows which variables are used by which methods at run time.

Let $|P|$ number of null intersections and $|Q|$ is number of non-empty intersection between sets of instance variables.

$$LCOO = |P| - |Q|, \text{ if } |P| > |Q|. \quad (1)$$

$$LCOO = 0 \text{ otherwise.}$$

Lower the value of LCOO denotes the higher cohesion. Higher the value of LCOO denotes that Components are difficult to maintain and reuse.

IV. Implementation

For implementation the data is taken from the telecom project [18]. The technologies used for the purpose are Python and aspect.

The steps of implementations are discussed below:

Firstly the installation of ADJT (AspectJ Development tools) is done which is followed by exploring the telecom project. For exploring the project two aspects are used: billing aspect and timing aspect. Billing aspect deals with billing details and the timing aspect is used to know the timing of connections, i.e. time at which the call is connected, disconnected and the duration of the call. Further the dynamic analyzer is designed to analyze the .java files and .aj files at run time. The analyzer is built in AspectJ for Aspect oriented programming. The analyzer gives the information about the functions, advices and the variables used at runtime. For analyzing the program at runtime pointcuts and advices are used. The pointcuts created for analyzing the program are described below:

```
pointcut myClass();
pointcut myConstructor(): myClass() && execution(new(..));
pointcut myMethod(): myClass() && execution(* *(..));
pointcut myGetAttribute(): get(* *) && myClass();
pointcut mySetAttribute(): set(* *) && myClass();
```

`myClass()` pointcut picks out each join point where the executing code is in telecom package. `myClass()` pointcut also ensure that all the function and variables to be used should be from the telecom package only. `myConstructor()` pointcut ensure that that all the function and variables to be used should be from the telecom package only and it picks join point whenever the constructor is executed. We know whenever a new object is created then its constructor is called automatically. `myMethod()` pointcut ensure that that all the function and variables to be used should be from the telecom package only and it picks the joinpoint whenever any function in telecom package is executed. That mean it is called whenever any function is executed in telecom package. `myGetAttribute()` pointcut ensure that that all the function and variables to be used should be from the telecom package only and it picks the joinpoint whenever any variable is accessed. `mySetAttribute()` pointcut ensure that that all the function and variables to be used should be from the telecom package only and it picks the joinpoint whenever any variable value is written or set. After writing the pointcuts we write advices to analyze the project at run time.

Advice defines crosscutting behavior. It is defined in terms of pointcuts.

The advices created are as follows:

```
before(): myConstructor()
after(): myConstructor()
before(): myMethod()
after(): myMethod()
before(): myGetAttribute()
after(): myGetAttribute()
before(): mySetAttribute()
after(): mySetAttribute()
```

With the help of above pointcuts and advices, the functions and variables at runtime are tracked easily. The tracked data is stored in a file named 'result.txt' by using `filewriter` and `bufferwriter` classes in java, for analysis at a later state.

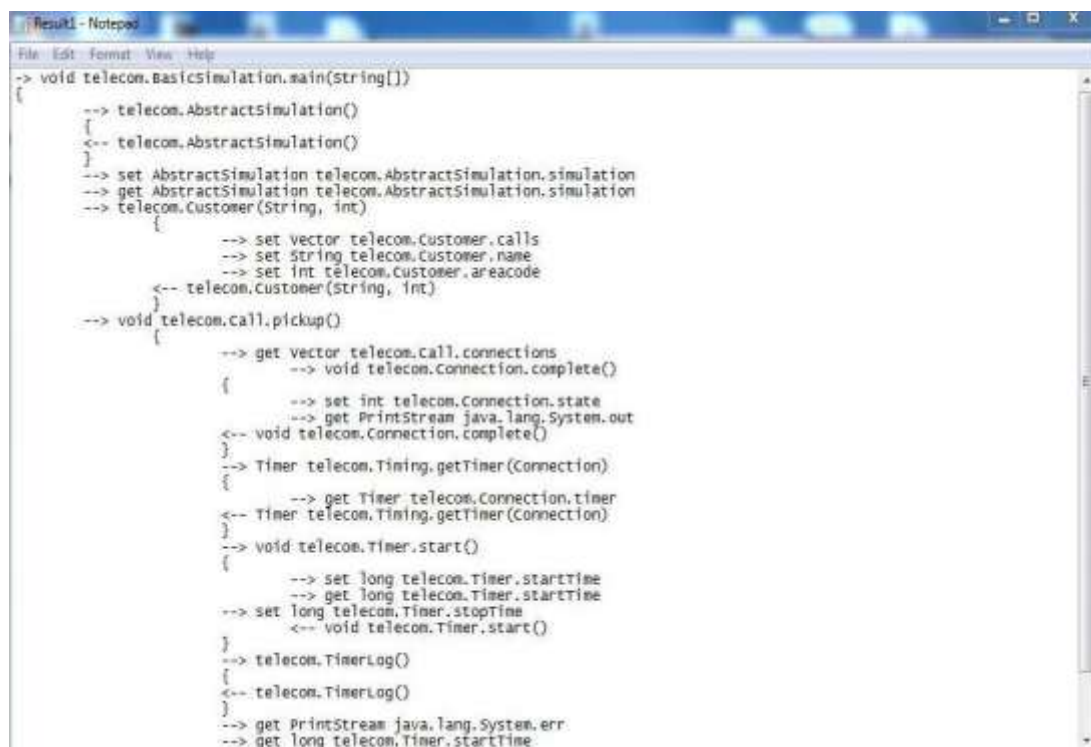
```
File f = new File("Result.txt");
```

```

if(!f.exists())
f.createNewFile();
FileWriter fw = new FileWriter(f.getName(),true);
BufferedWriter bw = new BufferedWriter(fw);
for( int i=1;i<depth;i++)
bw.write("\t");
bw.write("<-- ");
bw.write( str);
bw.newLine();
bw.close();

```

The file 'result.txt' is analyzed using python (figure 1.). The body of the result file starts with '{'. The mark '(' shows the name of the method which is called at run time. The body of the method is shown in '{' and '}' braces. The body also contains the record of the use of the variables in that method. If the value of the variable is accessed then it is represented by the 'get' keyword. The change of the value of the variable is shown with the 'set' keyword. The returning of the method with the mark '<--' followed by its name. All the information is shown with the indentation to make it easy to understand.



```

-> void telecom.BasicSimulation.main(string[])
{
    --> telecom.AbstractSimulation()
    {
        <-- telecom.AbstractSimulation()
    }
    --> set AbstractSimulation telecom.AbstractSimulation.simulation
    --> get AbstractSimulation telecom.AbstractSimulation.simulation
    --> telecom.Customer(String, int)
    {
        --> set vector telecom.Customer.calls
        --> set String telecom.Customer.name
        --> set int telecom.Customer.areaCode
        <-- telecom.Customer(String, int)
    }
    --> void telecom.Call.pickup()
    {
        --> get vector telecom.Call.connections
        --> void telecom.Connection.complete()
        {
            --> set int telecom.Connection.state
            --> get PrintStream java.lang.System.out
            <-- void telecom.Connection.complete()
        }
        --> Timer telecom.Timing.getTimer(Connection)
        {
            --> get Timer telecom.Connection.timer
            <-- Timer telecom.Timing.getTimer(Connection)
        }
        --> void telecom.Timer.start()
        {
            --> set long telecom.Timer.startTime
            --> get long telecom.Timer.startTime
            --> set long telecom.Timer.stopTime
            <-- void telecom.Timer.start()
        }
        --> telecom.TimerLog()
        {
            <-- telecom.TimerLog()
        }
        --> get PrintStream java.lang.System.err
        --> get long telecom.Timer.startTime
    }
}

```

Figure 1: result.txt file which contains sequence of execution of program.

To read the result file in order to count the number of variables called in a particular method, stack is used. File is read in a sequential manner. A line of the file can contain any of the following:

- Method call statement
 - --> void telecom.Call.pickup()
 - --> void telecom.Timer.start()
- Method return statement
 - <-- void telecom.Timer.start()
 - <-- void telecom.Call.pickup()
- Starting curling braces.('{')
- Returning curly braces.('}')
- Constructor call statement
 - --> telecom.Customer(String, int)
 - --> telecom.Billing()
- Constructor return statement
 - <-- telecom.Customer(String, int)
 - <-- telecom.Billing()

- Variable name with 'set' keyword.
 - --> set int telecom.Connection.state
 - --> set long telecom.Timer.startTime
- Variable name with the 'get' keyword.
 - --> get long telecom.Timer.startTime
 - --> get Timer telecom.Connection.timer

The push statement reads and puts every statement into the stack until the returning curly braces ('{') come. As soon as the returning curly braces are met, the elements are popped off from the stack. This stack parsing is done in the python.

Finally the value of LCOO is calculated dynamically by considering a class with n methods which refer to the instance variables. The number of time a variable is accessed is referred as Ni. To sum up the number of accesses the following method is used:

$$p^w = \sum_{1 \leq i, j \leq n} \{(Ni + Nj | Ii \cap Ij = \phi)\}$$

$$Q^w = \sum_{1 \leq i, j \leq n} \{(Ni + Nj | Ii \cap Ij = \phi)\}$$

Where $p^w = 0$, if $\{I_1\}, \dots, \{I_n\} = \phi$

V. Result and Discussion

The effects of dependent instance variables on cohesion metrics for Aspect -oriented programs have been investigated and an approach has been proposed to identify the dependency relations among instance variables.

A new well-defined dynamic cohesion measure has been proposed. 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 dynamic cohesion measures are better indicators of external software quality attributes such as change-proneness than the existing static cohesion metrics as proved by the experimental study.

For the object oriented programming cohesion metrics LCOM was developed in 1994 by C &K. It was the most popularly used metrics for the Cohesion. When the Aspect Oriented programming came in existence, an up gradation of this metrics of cohesion named LCOO was used. It was developed by the Sant Anna in 2003. But this was the static cohesion metrics.

The main difference between static and dynamic cohesion metrics is the scope of measurement at which cohesion is being measured. The scope of static cohesion measurement is always the whole class. On the other hand, scope of dynamic cohesion measurement can even be specific to a single object belonging to a class at run-time. In case of dynamic metrics, cohesion is first measured at object level and then cohesion of corresponding class is obtained by aggregating cohesion values of all objects belonging to that class. Moreover, static cohesion metrics attempt to predict the potential interactions that would take place at run-time, whereas dynamic cohesion metrics measure what is actually happening at run-time rather than predicting.

The implemented metrics is the dynamic cohesion metrics which is the extension of the LCOO metrics. There are two parameters P and Q which are used in the LCOO. In the work values of these Parameters are calculated dynamically. These values are obtained on the telecom project provided by the AspectJ plug-in in Eclipse. The result obtained is shown in the following figure 2. From the table 1, one can see that LCOO values for some classes and aspects are high and for some are low values. The classes and aspects with high LCOO represents that there is low cohesion for the respective classes or aspects and vice versa. The dynamic LCOO values are calculated by subtracting Q values from the P values, which are shown in table 1.

Classes and Aspect	Dynamic LCOO
AbstractSimulation	4
Billing	8
BillingSimulation	7
Call	0
Connection	0
Customer	0
Local	2
LongDistance	2
Timer	0
Timing	16

Table1. Dynamic LCOM values for Telecom Project

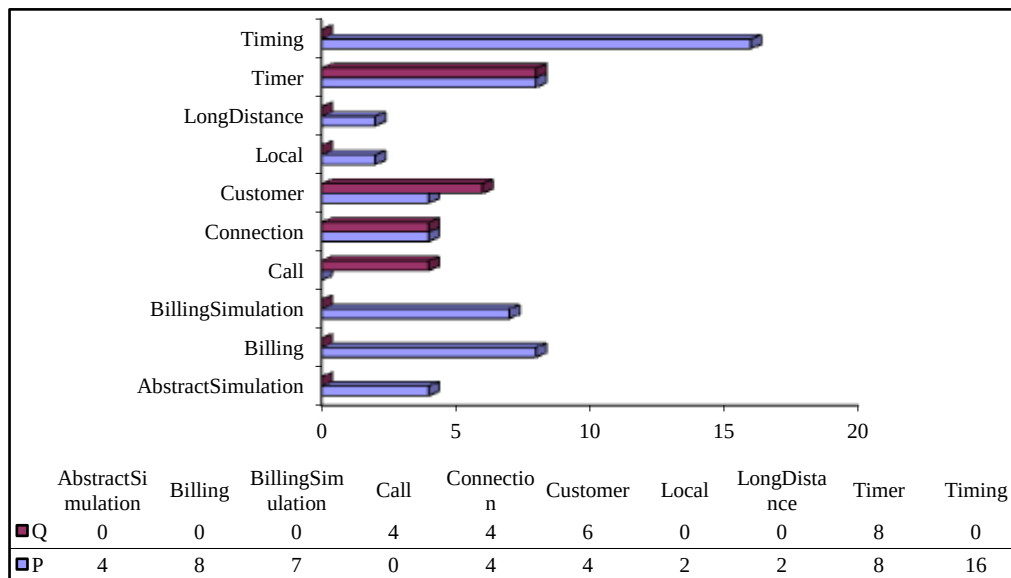


Figure 2: Run time values of P and Q for each class and aspect.

VI. Conclusion

In this research work dynamic and static cohesion metrics are studied. The effects of instance variables to calculate cohesion are also studied. Dynamic cohesion provides the realistic view of the software. Dynamic cohesion metrics reflect the true design of system. This work shows the cohesion between the modules which got called at run time.

References

- [1] J.M. Bieman and B.K. Kang, "Cohesion and reuse in an object-oriented system," Proceedings of the Symposium on Software Reusability, Seattle, WA, pp. 259–262, April 1955.
- [2] L.C. Braind, J. Daly and J. Wusr, "A unified framework for cohesion measurement in object-oriented system", Empirical Software Engineering, pp.67–117, 1998.
- [3] J. Bansiya and C.G. Davis, "A Hierarchical Model for Object- Oriented Design Quality Assessment," IEEE Trans. Software Eng., vol. 28, no. 1, pp. 4-17, Jan. 2002.
- [4] K. El-Emam and K. Melo, "The Prediction of Faulty Classes Using Object-Oriented Design Metrics," NRC/ERB-1064, vol. 43609, Nov. 1999.
- [5] T. Gyimo'thy, R. Ferenc, and I. Siket, "Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction," IEEE Trans. Software Eng., vol. 31, no. 10, pp. 897-910, Oct. 2005.
- [6] T.-S. Quah and M.M.T. Thwin, "Application of Neural Networks for Software Quality Prediction Using Object-Oriented Metrics," Proc. 19th IEEE Int'l Conf. Software Maintenance, pp. 116-125, Sept. 2003.
- [7] F. Brito e Abreu and M. Goulao, "Coupling and Cohesion as Modularization Drivers: Are We Being Over-Persuaded," Proc. Fifth European Conf. Software Maintenance and Reeng., pp. 47-57, 2001.
- [8] J.I. Maletic and A. Marcus, "Supporting Program Comprehension Using Semantic and Structural Information," Proc. 23rd IEEE Int'l Conf. Software Eng., pp. 103-112, May 2001.
- [9] S. Chidamber, D. Darcy, and C. Kemerer, "Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis," IEEE Trans. Software Eng., vol. 24, no. 8, pp. 629-639, Aug. 1998.
- [10] N. Gupta, P. Rao, Program execution based module cohesion measurement, in: Proc. 16th International Conference on Automated on Software Engineering (ASE '01), San Diego, USA, 2001.
- [11] A. Mitchell, 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.
- [12] A. Mitchell, J.F. Power, Run-time cohesion metrics: an empirical investigation, in: Proc. the International Conference on Software Engineering Research and Practice, pp. 532–537, 2004.
- [13] Varun Gupta, Jitender Kumar Chhabra, Dynamic cohesion measures for object-oriented softwar, Journal of Systems Architecture, pp. 452-462, 2011.
- [14] Jean-Francois Gelinas, Mourad Badri, Linda Badri, "A Cohesion measure of Aspects", Journal of object technology, vol 5, no 7, pp.75-95, sept-Oct 2006.
- [15] Martin Hitz, Behzad Montazeri, "Chidamber & Kemerer's metrics suite: a measurement theory perspective", IEEE Transactions on software engineering, vol 22, No. 4, pp. 267-271, April 1996.
- [16] H. Aman, T. Yanaru, M. Nagamatsu, K. Miyamoto, A metric for class structure complexity focusing on relationships among class members, IEICE Transactions on Information and System E81-D (12), pp. 1364–1373, 1998.
- [17] E. Arisholm, Dynamic coupling measures for object-oriented software, in: Proc. 8th IEEE Symp. on Software Metrics (METRICS'02), Ottawa, Canada, pp. 33–42, 2002.
- [18] AspectJ Home Page.- <http://eclipse.org/aspectj>
- [19] Jianjun Zhao, Baowen Xu, "Measuring Aspect Cohesion", 7th international Conference, FASE 2004, pp.54-68, April 2004.