Tool for Measuring Coupling in Object-Oriented Java Software

Mr. V. S. Bidve#1, Dr. P. Sarasu#2
[1]Ph.D. Scholar, [2]Director R & D
[#]Veltech Dr. RR & Dr. SR Technical University, Avadi, Chennai, India.
[1]vijay.bidve@gmail.com
[2]sarasujivat@gmail.com

Abstract-The importance of object-oriented software metrics is increasing day by day to evaluate and predict the quality of software. Coupling is one of the object-oriented metrics. It is a dependency degree to which one program module depends on one of the other modules. Coupling measures play a significant role in the quality aspect of object-oriented software, from design up to maintenance. To correctly predict the quality factors of object oriented software, the coupling should be accurately measured. In the related literature, we find many techniques to measure coupling. But, No any author explained the implementation of his technique(s) in details and made the tool available to know how exactly coupling has been measured.

In this paper, we propose a tool for measurement of coupling among classes of Java software. Java source code is taken as an input for the tool to measure coupling. The input Java code is parsed, and tokens are extracted. These tokens along with the code are used to measure different types of Coupling in Java software. Coupling of different sample Java codes is measured with the tool to observe values of each coupling type.

Keywords-Coupling, object-oriented software, metrics, measurement, class.

I. INTRODUCTION

Coupling measurement is used to evaluate the quality of software from design phase up to the implementation phase. This paper presents a tool that analyzes the Java source code to measure coupling amongst various modules of Java software. The coupling can be obtained from design document before implementation or from source after implementation [1]. The coupling measurement derived before implementation may be used in project planning, and implementation. Coupling measurement obtained from implementation can be used to reflect changes during implantation, and allowing the measurement to be more precise [1]. This research project addresses post implementation coupling measurement.

The authors A. Kavitha et al. [2], Gurunadha Rao Goda et al. [3], Arisholm et al. [4] proposed techniques of dynamic coupling measurement. Although, dynamic coupling measurement can capture some dependencies that static coupling measurement overlooks. The dependencies they capture are not complete because those are dependent on particular executions used [1], so we cannot rely on their tool completely. The author Jeff Offutt [1] has developed a static analysis tool named JCAT in 2008 to measure coupling by Java source code analysis. We contacted the author to Jeff Offutt [1] to know more about the working of the tool, but as per his reply, the tool is no more available. We couldn't find a tool in the literature which is comprehensive and reliable. Finally, we decided to develop a tool for static analysis of Java source code. The tool presented in the article is purely static; i.e. it analyzes the input Java source code without executing the same. This tool focuses on inter-class coupling rather than intra-class coupling. The tool is named as **J**ava **C**oupling **M**easurement **T**ool (JCMT).

The remainder of this paper is organized as follows. Section 2 describes the detailed review of related work conducted on various available coupling measurement techniques (tools). Section 3 summarizes the number of coupling measures (types) considered in the implementation. Section 4 provides the ways of analysis of Java code to measure couplings. Section 5 describes the implementation and working details of an automated tool developed for coupling measurement. Section 5 also presents the results obtained using JCMT tool with graphs in the form of bar-charts. Section 6 concludes the paper and confers about future work.

II. RELATED WORK

Coupling measurement has been attempted by various authors and numbers of tools are proposed in the literature. In this section, we are discussing the existing techniques (tools) of coupling measurement proposed by various authors. Our focus is to study existing coupling measurement tools rather than the available coupling metrics.

In 2002, Jarallah et al. [5] presented a tool for measuring inheritance coupling for object-oriented systems. The tool supports object-oriented languages such as Java, C++, and Smalltalk. The tool parses object oriented systems to collect inheritance coupling metrics data. The data is then abstracted in language independent format

and stored in system's database. Finally, query engines are applied on the database to calculate different coupling metrics. The authors have developed Java parser for software implemented in Java. As per the authors, other Object Oriented language parsers can be implemented without changes to the system. The main problem with the systems is that it calculates only inheritance coupling.

In 2004, Arisholm et al. [4] proposed a tool for dynamic coupling measurement in object-oriented systems. The authors have described two approaches for collecting coupling data. In the first approach, the coupling data is collected by executing the source Java programs. The tool utilizes interfaces provided by the Java Virtual Machine to collect message traces. The information such as caller method class, callee methods class, method invocation between classes, etc. is collected by a tool. It means, the first approach suggested by the author collect dynamic coupling only by message traces i.e. coupling other than method invocations and message passing is not under consideration by the tool. The second approach suggests collecting dynamic coupling data through analysis of dynamic UML models. But, as per author itself coupling obtained using UML interaction diagrams will tend to be underestimated because distinct elements of a set of possible messages will not be distinguishable with UML diagrams.

In between 2003 to 2007, Mitchell et al. [6], [7], [10], [20] described a technique for dynamic class level coupling measurement for runtime evaluation of a Java programs. Authors used the Java Platform Debug Architecture (JPDA) for implementation of coupling measurement tool. Using the tool each of the class is executed, and dynamic profile of the program in the form of occurrences of object instantiations, method calls, and instance variable accesses is collected to calculate coupling. As per the author itself, this tool is very time-consuming to generate a profile for the large application, and it is difficult to conduct an object-level analysis [7]. In the later implementation of the tool, the authors used Apache Byte Code Engineering Library (BCEL) [7]. This time, authors used to give an index to each method and field of a class file. When the application is executed, each probe records a "hit" in another class file. This run-time information is then utilized to calculate coupling measures. These tools described by the authors are limited to the measurement of runtime CBO coupling. Other couplings metrics are not considered for measurement by the authors.

In 2008, A. Kavita et al. [2] have proposed the technique of dynamic coupling measurement. As per the authors, the source code is introspected; all the functions are added with some trace events. Then the source code is compiled and allowed to run. During run time the actual function calls are extracted. Dynamic function calls from the source code are filtered out using standard coupling technique to get the dynamic coupling. As per the authors, the users have to add the functions to the trace events in the tool. Traces only capture the coupling of the code which comes under the execution of a particular event. Also, to select functions for the trace is an overhead to the users. In short, this tool is not fully automated, and multiple executions of the source code are required to capture all dimensions of coupling.

In 2008, Jeff Offutt et al. [1] have described a static analysis tool to identify coupling amongst the classes of Java source code. The tool is named as Java Code Analysis Tool (JCAT) by the authors. JCAT accepts the absolute path name of Java source package as an input. ANTLR is used as a grammar generator, which is used to generate Java parser. Abstract Syntax Trees (AST files) are generated by the Java parser. The information about class definitions, variable definitions, variable uses, method definitions, parameters and parameter uses, and method calls for each class is maintained by AST files. The JCAT tool extracts information from AST files and stores in a database (Access Database). Finally, ASTs are processed, and coupling is calculated using SQL queries. The idea of the tool and explanation given regarding the tool is highly significant. But, the use of AST files, Access database, and ASCII file is an overhead in the tool. A regular data structure of Java can replace AST files and Access Database. Also, coupling metrics are stored either in a spread sheet or ASCII text files; both file formats could be if JTable of Java is used. JCAT is a significant tool to measure coupling amongst Java classes, but authors have made the tool heavy by using many unnecessary components.

In 2009, S. Husein et al. [8] described a tool named CCMETRICS for calculating coupling and cohesion in object-oriented software. The tool receives object-oriented software source code as an input. The tool uses ANTLR to construct grammar files. The input is parsed according to the language. The parser is based on grammar files. The keywords are extracted from parsed input and are used to calculate coupling. The tool proposed by the author is a much-generalized tool; a lot of efforts will be required to construct language specific grammar files. The changes required to switch from one language to other is not properly mentioned by the authors. Coupling metrics calculations using keywords is needed to be explained in details.

In 2011, N. Kayarvizhy et al. [9] proposed a generic framework for computing coupling amongst object-oriented software components using XML. The tool accepts object oriented source code as an input. The source code is parsed and converted into intermediate generic XML format. XML generic format is extracted by metrics calculator to compute required metrics. The tool is developed in Java and supports to Java and C# languages. As per authors, they used XML as an intermediate generic format because XML is portable and compatible with Java. But, XML converter takes a lot of efforts to convert parsed data into XML intermediate generic format. Rather, this can be achieved using simple language data structures. Conversion of parsed data

into intermediate generic XML format and again to extract data for computing from XML format is an overhead in this proposed tool.

In 2014, R. Geetika et al. [11] described an implementation for analysis of Java code using the architectural diagram named methodology followed. The tool accepts Java sample code as an input, which is processed by bytecode instrumentation module. Call logs are used for evaluation of Dynamic Metrics as per the tool. The implementation details given by the authors about the tool are very diminutive; hence, it is very difficult to comment on the same.

III. SELECTION OF COUPLING METRICS

Selection of coupling metrics is a tedious task because of multiple reasons. A lot of work has been done in the field of coupling measures (metrics) of the object-oriented systems. As there is no standardization in the field, every author has proposed a different set of coupling measures. This resulted in an overlapping amongst the means (way of being coupled) of couplings. Also, different names are used by the various authors to describe the same kind of coupling mechanisms. Hence, there is need to remove the overlapping amongst the measures. To eliminate overlapping amongst the existing measures, we have classified them by type of interactions and mechanisms used for the coupling. A detailed classification of existing coupling measures is provided in Table 1. Most frequently described measures of various authors are considered in the classification. We have considered almost possible mechanisms of coupling as described in Table 1. The coupling names proposed in the second column of Table 1 are used in the development of the tool described in section 5. These measures are applicable to any object-oriented programming language, but the tool developed for this research is specific to Java.

TABLE I Classification of Existing Measures on the Basis of Type Interactions and Coupling Mechanisms

Sr. No.	Coupling names used in this proposed work	Type of interactions	Coupling Mechanisms	Existing coupling measures for the same mechanisms
1	Parameter coupling	Method-Method,	Method of one class invokes method/passes parameter/passes message to methods of another class or to make a call to the constructor of another class.	CBO [12], [13]
				Interaction Coupling [14]
				MM [15], [16]
				IC_OM, IC_CM, EC_OM, EC_CM [17]
				Invocation Coupling [18]
				MI [19]
2	Inheritance coupling	Class-Class	One class is a superclass of another class (Inheritance).	CBO, RFC [13]
				Inheritance Coupling [14],[18],[19]
3	Global coupling	Method-Attribute/ Class-Attribute (Friend)	Method of one class can directly access parts of the internal structure, of another class method (friend). Also to access common, shared, non-local variables of another class.	Interaction Coupling [14]
				Friend Coupling [15], [16]
				AR[19]
4	Data Abstraction Coupling	Class-Method/ Class-Attribute	One class is used in the implementation of methods of another class. One class is the domain of the instance variable, the local variable of another class.	Component Coupling [14]
				CM,CA [16]
				MP, MR, DA [19]
5	Import Coupling	ALL	All type of coupling due to any import mechanism.	Import Coupling [16], [19]
6	Export Coupling	ALL	All type of coupling due to an export mechanism.	Export Coupling [16], [19]
7	External/file coupling	Sharing of global devices.	Sharing an external device like the printer, HDD, external file by the two classes.	External Coupling [1], [14]

IV. ANALYSIS OF JAVA SOURCE CODE TO FIND COUPLINGS

This section presents a thorough analysis of Java source code. After analysis, we will be able to measure various types of couplings described in Table 1. The focus of analysis of source code is to cover all factors responsible for coupling. This analysis is practical based and focusing on different mechanisms that constitute a coupling amongst classes of Java source code. We are analyzing every component of a class to find the existence of coupling and its type.

4.1 Occurrence of Parameter Coupling

The parameter coupling occurs when a method of one class invokes method(s) or calls constructor(s) of other classes. In Java, there are two types of method calls, implicit and explicit. An implicit constructor call is made when a variable of another class type is defined and instantiated in a class. Explicit calls are made through object instances and static calls. All these calls with and without parameter passing are considered in parameter coupling. Let **X** and **Y** are two classes **m()** is a method of class **X**, the following are types of method calls made by a class **Y** which constitutes parameter coupling.

1. **X x= new X();** // implicit constructor call
2. **x.m();**// explicit call through object reference
3. **X.m();** //explicit static call.

As coupling is an inter-class relationship, hence we are not considering the currents class's method/constructor calls in parameter coupling. If **my(),** is a method of class **Y** then following calls will not be considered for parameter coupling of current class **Y**.

1. **Y y=new Y()** //constructor call for same class
2. **this.my()** // call to the method of same class

The above two types of calls come under intra-class coupling and are not considered under parameter coupling. Intra-class coupling is out of the scope of this research.

4.2 Occurrence of Inheritance Coupling

If one class inherits to another class there is inheritance type of coupling between two classes. The object instance of an inherited class can access public members of the base class; hence, this is a strong type of coupling between the classes. Java forms Inheritance relationship using the keywords *extends* and *implements* with classes and interfaces respectively.

In this study, we are focusing on another angle of inheritance relationship. When one class **A** inherits another class **B,** then class **A** is coupled with class **B** through inheritance coupling. Similarly, here we are considering that; class **B** is also coupled to class **A** by the same coupling. We have decided to increase the coupling count of both the classes **A** and **B** if they are coupled via inheritance coupling.

In this research, we are considering only direct inheritance coupling not the indirect. In the example shown in figure 1, class **A** is directly coupled with classes **B**, **C**, and **D** and indirectly coupled with class **E**. Indirect coupling happens due to multilevel coupling between the classes. Indirect inheritance coupling may create unnecessary confusion and increases the complexity of the project. The direct-coupling analysis covers all associations between the classes by collecting each component of a class which is responsible for coupling. Hence, the indirect coupling is not required; so kept out of the scope of this project.

```
class A {
  public void methodA()  {
    System.out.println("method of class A");  } }
class B extends A {
  public void methodB()  {
    System.out.println("method of class B");  } }
class C extends A {
 public void methodC()  {
 System.out.println("method of class C");  } }
class D extends A {
  public void methodD()  {
    System.out.println("method of class D");  } }
class E extends D {
  public void methodD()  {
    System.out.println("method of class E");  } }
```

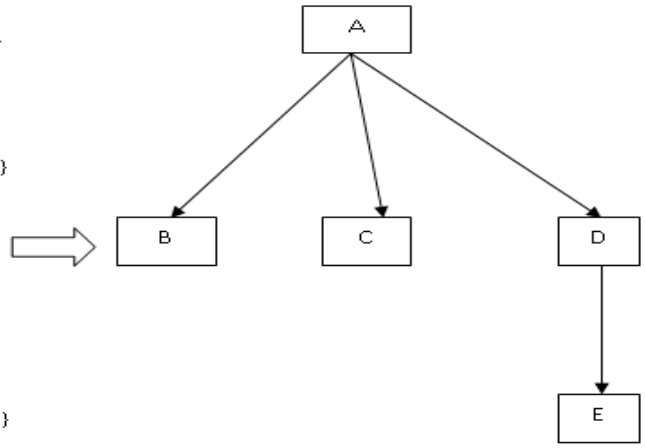


Fig. 1 Class Hierarchy and Inheritance

4.3 Occurrence of Global Coupling

Global coupling occurs due to the variables defined in one class and used in another class. The variables are particularly of the type public and protected. The public global variables are accessed using object instances, and static variables are accessed by static calls. In C++ private members of other class can be accessed via friend mechanism. As Java does not support friend mechanism hence references to private members are considered in this project.

4.4 Occurrence of Data Abstraction Coupling

Data abstraction coupling takes place when another class is used as a domain (data type) in the definition of current class. Another class can be used as a domain for the return type of method, in the definition of a method, or in the definition of the current class itself.

4.5 Occurrence of Import and Export Coupling

Import coupling of a class constitutes a coupling due to all types of references and invocations made by a class to the members of other classes. A class may import methods, global, static, or instance variables of another class. Use of another class as a domain is also a type of import.

Export coupling takes place when the members of a current class are being used in the implementation of another class. The methods, global, static, or instance variables of a current class may be exported. A class may be exported to use as a domain by another class.

4.6 Occurrence of External Coupling

Two or more classes are coupled via external coupling if they are referencing to the shared common devices like printer, HDD, or external file. An access to common shared devices by more than one class contributes to the external coupling between those classes as shown in figure 2. Total reference of a class to the shared devices in association with other classes leads to an external coupling.

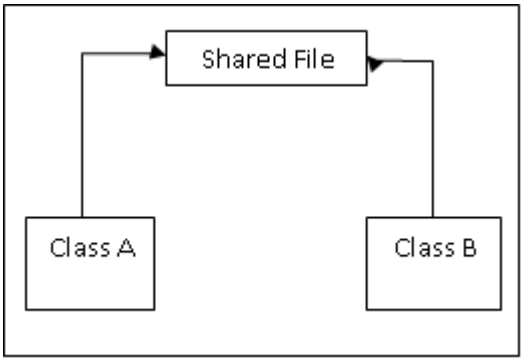


Fig. 2 External Coupling

An executable Java code is analyzed to verify the appearance of above coupling types. This analysis presents a systematic way of coupling measurement between the classes of Java source code.

V. A TOOL FOR MEASURING COUPLING

We have developed a Java Coupling Measurement Tool (JCMT) to measure coupling amongst classes of object-oriented Java software. The tool measures all types of couplings defined in Table 1 using the techniques described in section 4.

JCMT is a static source code analysis tool developed in Java. The tool comprises of several software modules including Java parser, method visitor, class visitor, coupling computation unit, etc. Figure 3 shows JCMT's architecture diagram, which tells the process flow of the tool.

The JCMT accepts .Java file as an input then uses **JavaParser** to verify the syntax of an input Java file and to converts it into **Compilation Unit**. Compilation Unit is a properly formatted ordered Java code generated by a **JavaParser.**

JCMT extracts information from compilation unit using **ClassVisitor** and **MethodVisitor** modules. The **ClassVisitor** visits the class header, attributes, fields, methods, inner information, and end of each class of an input Java source file. Similarly, the **MethodVisitor** visits the annotations, attributes, fields, local variables, and end of methods. The extracted data from compilation unit is stored in **ClassArrayList**. **Coupling Computation Unit** refers the data of **ClassArrayList** in association with class extraction module to compute coupling of each type described in Table 1 using techniques given in section 4. The computed couplings values are initially captured in the hash table and then displayed using the Java **JTables**.

As compared with the tools described in the literature this tool is lightweight and developed using a minimum number of inbuilt Java components.

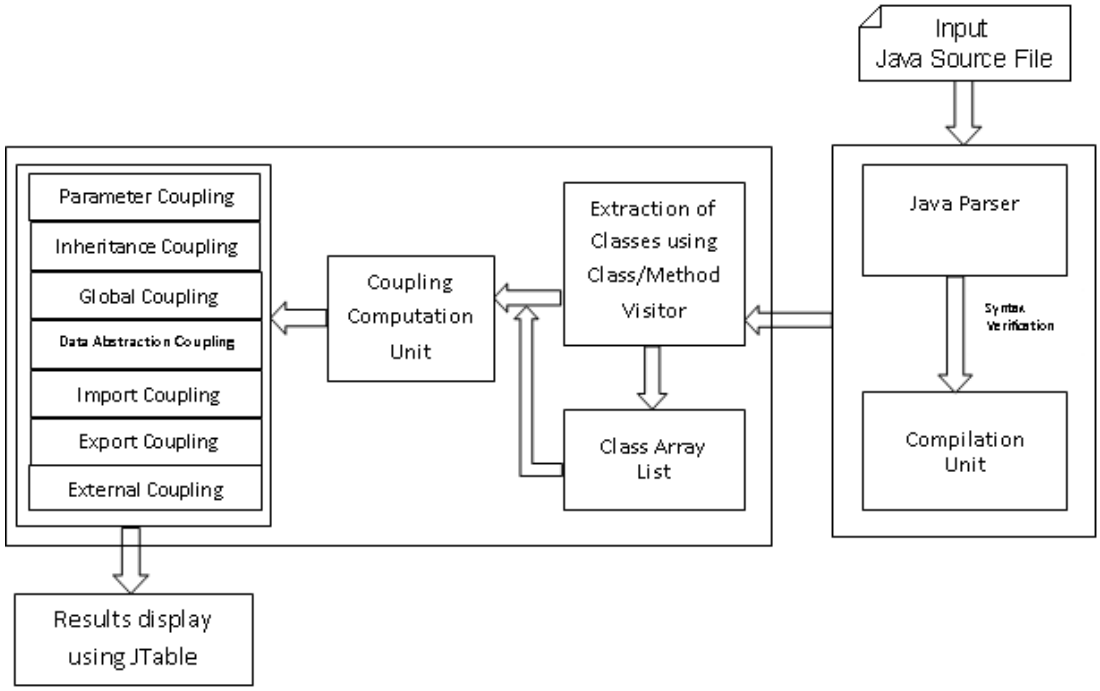


Fig. 3 Architecture Diagram of JCMT

5.1 JCMT user interface

Figure 4 shows JCMT's dashboard. Browse button is provided showing three dashes as a caption to accept Java input file. The path of selected Java file is shown in a text box. The **View File** button is used to opens an input Java file if the user wants to see the contents. The **Parse** button parses the input Java file and displays output in **JTable** by performing computations**.** If an input Java file contains any error; **Parser** displays an error message. **JTable** displays the output in the form of rows and columns. The row headers are the name of classes belonging to an input Java file, and the column headers are different coupling types whose value is calculated. Every row of the **JTable** displays coupling values of each type for the respective class. A nonzero value indicates the total number occurrences of a particular coupling type where the zero value is a nonexistence of that coupling type. At the end of dashboard the buttons **Average Graph**, **Coupling Graph**, and **Class Graph** are provided to show the coupling output in the form of bar charts as shown in figure 5. The dashboard can be closed directly using close window button.

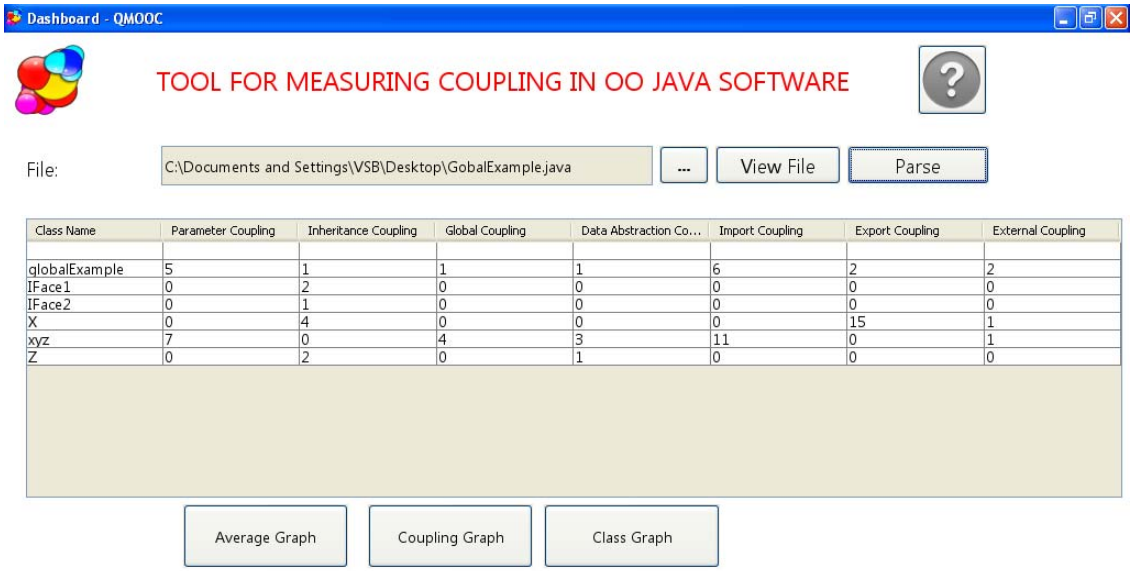


Fig. 4 User interface of JCMT tool

5.2 JCMT software design and implementation

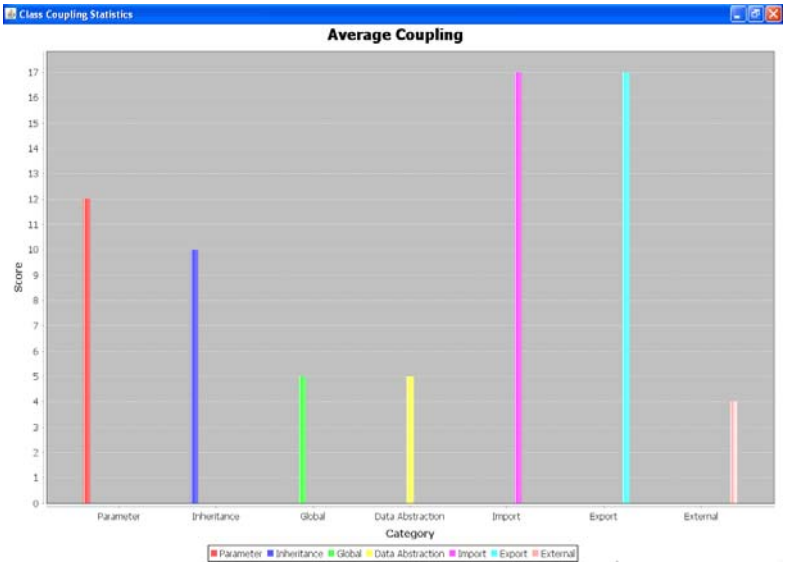
JCMT has two main packages, *coupling* and *japa* (Java Parser). The *coupling* package is responsible for metrics calculations, GUI, and graphs, where the *japa* package is responsible for parsing the input Java file and to convert it into a class array list.

JCMT calculates the coupling of each class with other classes by analyzing the class structure. The coupling definitions are the base for identifying various kinds of couplings using JCMT. JCMT is tested for the examples of different sizes, and the output is verified manually. Theoretically, there is no upper limit on the number of classes handled by JCMT for an input Java source file.

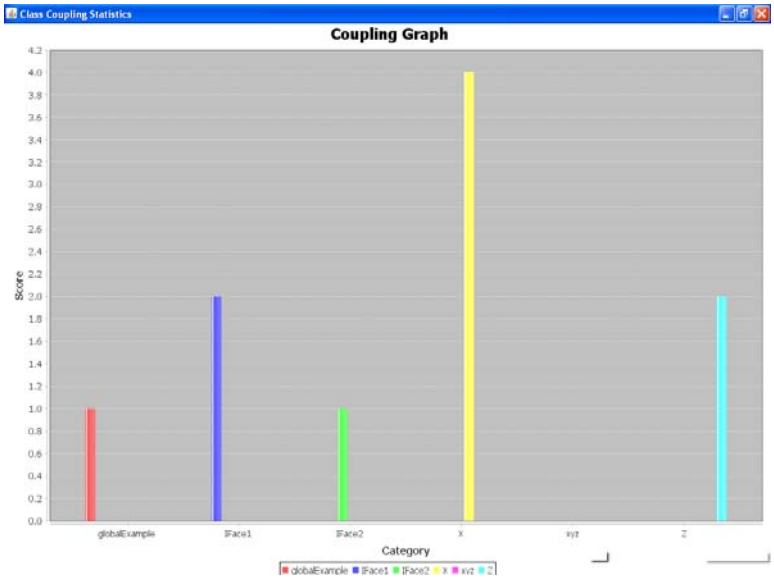
5.3 Results

We used JCMT to measure coupling of a sample Java source file named *GlobalExample* and results are displayed in the JTable shown in figure 4. *GlobalExample* is a sample Java file to test JCMT's results. Instead of *GlobalExample* we can consider any executable java file as an input to JCMT. The graphs of the output obtained from *GlobalExample* are shown in figure 5.

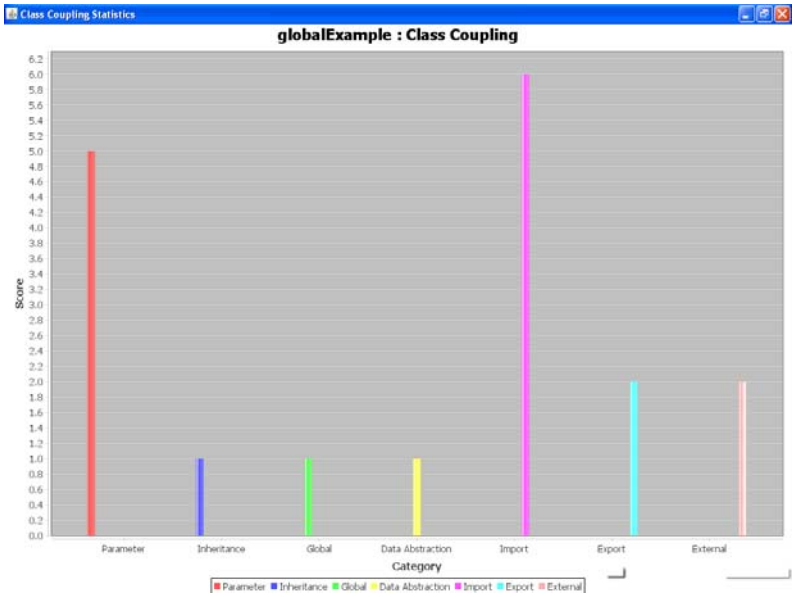
This paper only describes the implementation and the use of JCMT tool. In this paper, we are not commenting anything on the results as it is a future scope of this project.



(a) Average Graph



(b) Couplingwise Graph



(c) Classwise Graph

Fig. 5 Graphs of output

VI. CONCLUSION AND FUTURE WORK

This paper presents a tool for measuring coupling in object oriented Java software. We selected the coupling measures which are unambiguous and cover all aspects of class associations for the implementation of this tool. The techniques used in the coupling measurement of this tool are based on Java language features. The tool developed is lightweight and uses a minimum number of inbuilt Java components.

In the future, we planned to use this tool for measuring the coupling of large size source codes and the results will be used to find its impact on different quality parameters of object-oriented software.

REFERENCES

[1] Jeff Offutt, Aynur Abdurazik, and Steve Schach, “Quantitatively Measuring Object-Oriented Couplings”, Springer's Software Quality Journal, 6(4):489-517, December 2008.

[2] A. Kavitha and Dr. A. Shanmugam, "Dynamic Coupling Measurement of Object-Oriented Software Using Trace Events", In proc. of the 6th International Symposium on Applied Machine Intelligence and Informatics, 2008 (SAMI 2008), pp. 255-259, Jan. 2008.

[3] Gurunadha Rao Goda and Avula Damodaram, “Measurement of Dynamic Coupling in an Object Oriented System Based on Trace Events”, American J. Scientific Research, ISSN 1450-223, 10(7): 43-55, 2011.

[4] Erik Arisholm, Lionel C. Briand, and Audun Føyen, “Dynamic Coupling Measurement for Object-Oriented Software”, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 30, NO. 8, AUGUST 2004.

[5] AlGhamdi, J., Elish, M., and Ahemed , M. “A tool for measuring inheritance coupling in Object-Oriented Systems”, Special issue of the Software Science Journal, October 2001.

[6] A Mitchell, JF Power, “Runtime Coupling Metrics for the Analysis of Java Programs”, preliminary results from SPEC and Grand suites 2003.
[7] Mitchell Aine, James F. Power. "Using object-level run-time metrics to study coupling between objects", Proceedings of the 2005 ACM symposium on applied computing. ACM, 2005.
[8] Husein, Sukainah, Oxley Alan, “A Coupling and Cohesion Metrics Suite for Object-Oriented Software”, International Conference on Computer Technology and Development, vol.1, no., pp.421-425, 13-15 Nov. 2009.
[9] N. Kayarvizhy, S. Kanamani, “An Automated Tool for Computing Object Oriented Metrics using XML”, Proceedings of International Conference on Advances in Computing and Communication ACC2011, Springer, 2011, Vol. 191, pp. 69-79.
[10] Michael English, TonyCahill, JimBuckley, “Construct specific coupling measurement for C++ software”, Computer Languages, Systems & Structures 38 (2012), 300–319.
[11] Rani Geetika, Paramvir Singh, “Empirical investigation into static and dynamic coupling metrics”, ACM SIGSOFT Software Engineering Notes 39(1): 1-8 (2014).
[12] S.R. Chidamber, C.F. Kemerer, “Towards a Metrics Suite for Object Oriented design”, in A. Paepcke, (ed.) Proc. Conference on Object-Oriented Programming: Systems, Languages and Applications (OOPSLA’91), October 1991. Published in SIGPLAN Notices, 26 (11), 197-211, 1991.
[13] S.R. Chidamber, C.F. Kemerer, “A Metrics Suite for Object Oriented Design”, IEEE Transactions on Software Engineering, 20 (6), 476-493, 1994.
[14] J. Eder, G. Kappel, M. Schrefl, “Coupling and Cohesion in Object-Oriented Systems”, Technical Report, University of Klagenfurt, 1994.
[15] L. Briand, P. Devanbu and W. Melo, “An Investigation into Coupling Measures for C++,” Proc. 19th Int’, l Conf. Software Eng., ICSE ‘, 97, Boston, pp. 412-421, May 1997.
[16] Briand, Lionel C., John W. Daly, and Jurgen K. Wust, “A unified framework for coupling measurement in object-oriented systems.” Software Engineering, IEEE Transactions on 25.1 (1999): 91-121.
[17] Erik Arisholm, Lionel C. Briand, and Audun Føyen, “Dynamic Coupling Measurement for Object-Oriented Software”, IEEE Transactions on Software Engineering, Vol. 30, NO. 8, August 2004.
[18] Huan Li, “A Novel Coupling Metric for Object –Oriented Software Systems”, IEEE International Symposium on International Journal of Computer Applications (0975 – 8887) Volume 27– No.10, August 2011Knowledge Acquisition and Modeling Workshop, pp. 609-612, 2008.
[19] Husein, Sukainah, Oxley Alan, “A Coupling and Cohesion Metrics Suite for Object-Oriented Software”, International Conference on Computer Technology and Development, vol.1, no., pp.421-425, 13-15 Nov. 2009.
[20] A Mitchell, JF Power, “An empirical investigation into the dimensions of run-time coupling in Java programs”, In Third Conference on the Principles and Practice of Programming in Java (pp. 9-14) 2004. Las Vegas, Nevada, USA.

AUTHOR PROFILE

Author1 Mr. V. S. Bidve Ph.D. scholar in Veltech Dr. RR & Dr. SR Technical University, Avadi, Chennai, India. Also works as a professor. Completed M.Tech. in IT. Degree and having 14 years teaching experience.

Author1 Dr. P. Sarasu Working as a research dean in Veltech Dr. RR & Dr. SR Technical University, Avadi, Chennai, India. Also working as a Ph.D. supervisor. Having 20+ years experience in teaching.