Interaction Coupling: A Modern Coupling Extractor

S. Gomathi and P. Edith Linda

Abstract Software development plays a vital role in interaction between the methods, classes, and attributes. Coupling is one of the most vibrant internal qualities to measure the design performance. Many object-oriented metrics have been proposed to evaluate different aspects of object-oriented program using coupling. This paper presents a new modern approach, which depicts the concept of interaction coupling, and a prototype is developed to measure the interaction coupling. Three types of metrics response for class (RFC), message-passing coupling (MPC), and method invocation coupling (MIC) that may invoke methods are analyzed, measured, and summarized.

Keywords Interaction coupling · Class loader · Extractor · Reliability · Efficiency

1 Introduction

Object-oriented development has proved its value for systems that must be maintained, reused, and modified. Coupling has been defined as one of the most important qualitative attributes to measure the performance of software at design or implementation phase [1]. Coupling can be categorized into three types: component coupling, interaction coupling, and inheritance coupling. This research is mainly focused on interaction couplings such as MPC, RFC, and MIC. Interaction coupling occurs when the methods of a class invoke methods of another class. In this, the main thing that is to be understood is about message. A message

S. Gomathi (✉)
Sri Krishna Arts and Science College, Coimbatore, India
e-mail: gomathisrinivasan88@gmail.com

P. E. Linda
Dr. G. R. Damodaran College of Science, Coimbatore, India
e-mail: p.lindavinod@gmail.com

G. S. S. Krishnan et al. (eds.), *Computational Intelligence, Cyber Security and Computational Models*, Advances in Intelligent Systems and Computing 246,
DOI: 10.1007/978-81-322-1680-3_26,

is a request that an object makes of another object to perform an operation [2]. The operation executed as a result of receiving a message is called a method.

The rest of the paper is organized as follows. Section 2 summarizes about the metrics used in the previous papers. Section 3 highlights about the problem with existing coupling parameters. Section 4 highlights the types of interaction coupling, and Sect. 5 depicts the framework. Section 6 summarizes a new algorithm design to measure those coupling parameters. Sections 7 and 8 offer the results and the conclusion.

2 Literature Survey

Li and Henry [3] identified a number of metrics that can predict the maintainability of a design. They define message-passing coupling (MPC), defined as the number of send statements defined in a class. The number of send statements that are sent out from a class may indicate how dependent the implementation of the local methods is on the methods in other classes. MPC only counts invocations of methods of other classes, not its own. Chidamber and Kemerer [2] proposed and validated a set of six software metrics for object-oriented systems, including two measures for coupling RFC and CBO. The response set (RS) of a class is a set of methods that can potentially be executed in response to a message received by an object of that class. A given method is counted only once. RFC includes methods called from outside the class and also a measure of the communication between that class and other classes. Vijaya Saradhi and Sastry [4] proposed a unique new approach for metric, which delivers the system quality based on cohesion and coupling between classes. The proposed metric shows the relationship between the classes based on the flow in control and the number of occurrences of class, which is mainly based on the existing systems' input. Arisholm et al. [5] stated that the relationships between coupling and external quality factors of object-oriented software have been studied extensively for the past few years. The authors concluded about the empirical relationships between class-level coupling, class fault- proneness and to measure coupling is through structural properties and static code analysis.

3 Problem Specifications

The current research on modeling and measuring the relationships between object-oriented programs through coupling analysis is insufficient. Coupling measures are incomplete in their precision of definition and quantitative computation. Moreover, some existing coupling measures do not reflect the differences in and the connections between design-level relationships and implementation-level connections. Hence, the way the coupling is used to solve problems is not satisfactory. Measuring various types of coupling manually is not possible, and tools fail to measure some important coupling parameters.

4 Proposed Metrics

Different types of coupling have evolved over time. But single type of coupling is inadequate to reduce the complexity of the code. This paper compared various couplings and finally selected the best coupling parameters to evaluate the complexity, quality factor, and reliability of object-oriented program [6]. We propose three metrics MPC, RFC, and MIC that help to detect the reusability and efficiency in design of object-oriented programs at the early stage.

4.1 Message-Passing Coupling

MPC is the count of total number of functions and procedure calls made to external units [3]. The MPC measures the dependency of local methods to methods implemented by other classes.

Viewpoints: This allows for conclusions on reusability, maintenance, and testing effort [7]. Message passing is calculated at the class level.

$$\mathrm{MPC} = \sum_{j=1}^{n} \mathrm{MC}_e \tag{1}$$

where MC_e is method call to external class and j is the number of classes.

4.2 Method Invocation Coupling

It is defined as the relative number of classes that receive the message from the particular class [8].

Viewpoints: The number of methods invoked implies the program reliability and efficiency [9]. The methods invoked should be minimum so as to maintain the system throughput.

$$\mathrm{MIC} = \sum_{j=1}^{n} \mathrm{MI}_i \tag{2}$$

where MI_i is method invoked from other classes and j is the number of classes.

4.3 Response for Class

RFC is the number of functions and procedures that can be potentially be executed in a class. Specifically, RFC is the number of operations directly invoked by member operations in a class plus the number of operations themselves [2].

Viewpoints: If the larger number of methods can be invoked in response to a message, the testing and debugging of class becomes more complicated [10].

$$\mathrm{RFC} = \sum_{j=1}^{n} \mathrm{MC}_e + \mathrm{MC}_I \tag{3}$$

where MC_e is method call to external class, MC_I is methods of its own class, and j is the number of classes.

5 Framework of the Proposed System

The proposed framework is named as interaction coupling extractor (ICE), which is depicted in Fig. 1.

The framework has been partitioned into three phases. Each phase will perform some unique and important tasks.

5.1 Input Phase

This is the initial phase which is used to get input from the user. Programmer should give the jar file as input. Java ARchive (JAR) file is a collection of text, images, packages, and class files of java. A JAR file is essentially a zip file that contains an optional META-INF directory. But jar file cannot be measured directly. It must be extracted into individual class files. Jar files of variable size are given as input to interaction coupling extractor.

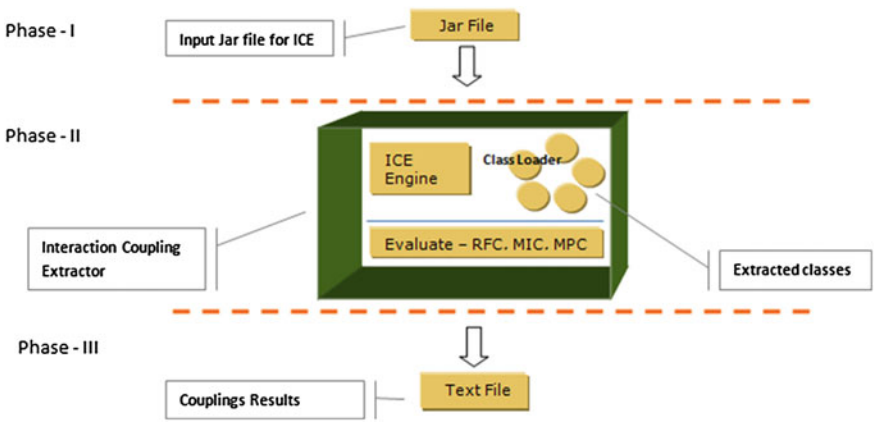

Fig. 1 Interaction coupling extractor

5.2 Processing Phase

This is the second level of extraction. Once a jar file is accepted, the file will be sent to interaction coupling extractor engine where the engine will separate the jar file into individual class files. That class files will be loaded into class loader in order to evaluate the interaction coupling parameters (RFC, MPC, and MIC).

5.3 Result Phase

The measured class will be displayed in the text file, which shows the number of RFC, MIC, and MPC measures of individual class. Then, the results are analyzed.

6 Algorithm Design

This design algorithm shows how to design the interaction coupling parameters MPC, RFC, and MIC and also shows how to measure those parameters.

Algorithm - Interaction Coupling Extractor (ICE)

Task: *To measure the relationship between methods of a jar file*

Input: *Jar file with variable size*

Output: *the Measured value of MPC, MIC and RFC*

Begin

Terminologies: *MC – Methods called, MI – Methods Invoked, MI_i – Methods Invoked from other class, MC_e – Methods call to external class, MC_I – Methods of its own class*

Calculate Message Passing Coupling using

$$MPC = \sum_{j=1}^{n} MC_e$$

Calculate Response for Class

$$RFC = \sum_{j=1}^{n} MC_e + MC_I$$

Calculate Method Invocation Coupling

$$MIC = \sum_{j=1}^{n} MI_i$$

The measured parameters are tabulated in a text file

End

Table 1 Results of JavaCSV.jar

Class name	RFC	MPC	MIC
com.csvreader.CsvReader$HeadersHolder	2	1	1
com.csvreader.CsvReader$RawRecordBuffer	1	0	1
com.csvreader.CsvWriter	58	30	0
com.csvreader.CsvWriter$UserSettings	1	0	1
com.csvreader.CsvReader$UserSettings	1	0	1
com.csvreader.CsvReader	86	36	0
com.csvreader.CsvWriter$Letters	1	0	0
com.csvreader.CsvReader$DataBuffer	1	0	1
com.csvreader.CsvReader$ColumnBuffer	1	0	1
com.csvreader.CsvReader$Letters	1	0	0
com.csvreader.CsvReader$StaticSettings	1	0	0
com.csvreader.CsvReader$ComplexEscape	1	0	0

7 Results

Two jar files, namely JavaSCV and JEdit, are given as input to interaction coupling extractor (ICE), and the results are shown below.

File Name: JavaCSV.jar
Size: 14.0 Kb
Source: findjar.com/jar/net/sourceforge/javacsv/javacsv/2.0/javacsv-2.0.jar.htm
Number of classes: 12

From Table 1, it is clearly shown that the RFC and MPC for class com.csvreader.CsvReader are higher. Hence, that class must be reprogrammed in order to make the jar file efficient.

File Name: JEdit.jar
Size: 114 K
Source: http://www.java2s.com/Code/Jar/j/Downloadjeditsyntaxjar.htm
Number of classes: 46

From Table 2, it is clearly shown that the RFC and MPC for class installer.SwingInstall are higher. MIC of installer.OperatingSystem class is high. Hence, both the classes must be reprogrammed in order to make the jar file efficient.

Other jar files, namely JUnit.jar, HSQL.jar, with more than 200 class files, are given as input, and the results are analyzed. The efficiencies of all the classes are measured. Maintainability of the program is improved based on the measurements. The classes that exceed the maximum ranges must be reprogrammed in order to make the program efficient, to reduce complexity, and to make it more flexible.

Table 2 Results of JEdit.jar

Class name	RFC	MPC	MIC
installer.CBZip2InputStream	36	6	1
installer.CBZip2OutputStream$1	−1	−2	0
installer.CBZip2OutputStream$StackElem	2	1	1
installer.CBZip2OutputStream	43	10	0
installer.CRC	5	1	2
installer.ConsoleInstall	36	34	1
installer.ConsoleProgress	10	4	2
installer.Install	45	38	9
installer.InstallThread	33	30	3
installer.InvalidHeaderException	3	2	2
installer.NonInteractiveInstall	26	25	1
installer.OperatingSystem$HalfAnOS	5	3	1
installer.OperatingSystem$MacOS	6	3	1
installer.OperatingSystem$OSTask	13	4	8
installer.OperatingSystem$Unix$ManPageOSTask	16	13	1
installer.OperatingSystem$Unix$ScriptOSTask	20	17	1
installer.OperatingSystem$Unix	25	19	4
installer.OperatingSystem$VMS	6	4	1
installer.OperatingSystem$Windows$JEditLauncherOSTask	14	11	0
installer.OperatingSystem$Windows	7	4	1
installer.OperatingSystem	15	10	10
installer.ServerKiller	25	22	1
installer.SwingInstall$ActionHandler	5	3	1
installer.SwingInstall$ChooseDirectory$1	4	2	1
installer.SwingInstall$ChooseDirectory$ActionHandler	14	12	1
installer.SwingInstall$ChooseDirectory	30	27	2
installer.SwingInstall$DirVerifier$1	7	5	1
installer.SwingInstall$DirVerifier$2	8	6	1
installer.SwingInstall$DirVerifier	35	25	3
installer.SwingInstall$SelectComponents	37	33	1
installer.SwingInstall$SwingProgress$1	3	1	1
installer.SwingInstall$SwingProgress$2	4	2	1
installer.SwingInstall$SwingProgress$3	3	1	1
installer.SwingInstall$SwingProgress$4	5	3	1
installer.SwingInstall$SwingProgress$5	3	1	1
installer.SwingInstall$SwingProgress	19	12	1
installer.SwingInstall$TextPanel	16	15	1
installer.SwingInstall$WindowHandler	3	1	1
installer.SwingInstall$WizardLayout	17	11	1
installer.SwingInstall	70	66	4
installer.TarBuffer	34	18	2
installer.TarEntry	63	32	4
installer.TarHeader	22	13	1
installer.TarInputStream$EntryAdapter	7	3	0
installer.TarInputStream	41	25	1
installer.TarOutputStream	33	21	0

8 Conclusion and Future Scope

This paper introduced a framework for interaction coupling for object-oriented systems. The interaction coupling is best suited to find the reusability and efficiency of the object-oriented systems. The algorithm used to implement the concept of RFC, MPC, and MIC is easy to understand. The detailed result sets show how the coupling parameters are measured and evaluated. The future work is to measure the component and inheritance coupling. The component coupling show how the arguments passed from one method to another method are measured. Inheritance coupling is used to measure the coupling between the inherited classes. The proposed framework will measure both types of coupling, and the measured parameters are displayed as chart.

References

1. Amjan Shaik et al., "Metrics for Object Oriented Design Software Systems: A Survey", Journal of Emerging Trends in Engineering and Applied Sciences ISSN: 21417016 (2010).
2. Chidamber, Shyam and Kemerer, Chris, "A Metrics Suite for Object-Oriented Design", IEEE Transactions on Software Engineering, June, (1994), pp. 476–492.
3. Li and Henry, "Object Oriented Metrics which predict maintainability", Journal of Systems and Software, Volume 23 Issue 2, Nov (1993).
4. L.C. Briand, J.W. Daly and J.K. Wust, "A Unified Framework for coupling measurement in Object Oriented Systems", IEEE Transaction on Software Engineering (1999), Vol. 25, Issue.
5. E. Arisholm et al., "Dynamic coupling measurement for object-oriented software", IEEE Transactions on Software Engineering, vol. 30, pp. 491–506, August 2004.
6. M.V. Vijaya Saradhi and B.R. Sastry "ESPQ: A New Object Oriented Design Metric for Software Quality Measurement" International Journal of Engineering Science and Technology (2010), Vol. 2, Issue. 3.
7. Harrison. R et al, "Coupling metrics for object-oriented design" proceedings for the Fifth International Software Metrics Symposium, (1998), pages 150–157.
8. Ramanath Subramanyam et al., "Empirical analysis of CK metrics for Object-Oriented design complexity: Implications for Software Defects", IEEE transactions on software engineering (2003), vol. 29, No. 4.
9. K K Agarwal, Yogesh Singh, ArvinderKaur and Ruchika Malhotra, "Empirical Study of Object-Oriented Metrics", (2006), vol. 5, No. 8.
10. Magnus Andersson, Patrik Vestergren, "Object-Oriented Design Quality Metrics" http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.1.5047.