International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970)
Volume-1 Number-2 Issue-2 December 2011

A Novel Java Based Computation and Comparison Method (JBCCM) for Differentiating Object Oriented Paradigm Using Coupling Measures

Sandeep Singh[1], Geetika S.Pandey[2], Yogendra Kumar Jain[3]
M.Tech, Department of Computer Science & Engineering, SATI Vidisha, M.P., India[1]
Assistant Professor, Department of Computer Science & Engineering, SATI Vidisha, M.P., India[2]
HOD, Department of Computer Science & Engineering, SATI Vidisha, M.P., India[3]

Abstract

In this paper we propose a novel java based computation and comparison method (JBCCM). In this method we taking three type of object oriented files for showing the computation. Those three files belong to C++, Java and C#. We first compute class, Inheritance, Interface, object and Line of Codes (LOC). Then we assume three databases based on several properties of C++, java, C#. Then we compare three files Based on class (BOC), Based on Inheritance (BOI), Based on Interfaces (BOIN), Based on Object (BOO) and Based on LOC (BOL). Then we deduce a comparative result for that particular object oriented file. The result approximate that the file is best suited on which platform. So we can deduce best platform and coupling measures for Object Oriented Paradigm.

Keywords

OOP, JBCCM, C++, Java, C#

1. Introduction

While the primary purpose of Object Oriented Programming (OOP) is to improve program comprehension and ease system maintenance, empirical evidence exists to suggest that use of inheritance can have the opposite effect (Harrison et al. 2000, Cartwright 1998, Cartwright and Shepperd 2000). The question; does OOP improve system comprehension and maintainability? Therefore remains unanswered. As a SE community, we know very little about the effect of inheritance and its limitations from a maintainability perspective. Previous studies suggest that inheritance should be used with care and only when necessary (Wood et al. 1999). In addition, we expect systems to evolve due to the changes of requirements and/or the environment in which they are operating. Previous studies have analyzed software systems from a maintenance and evolution perspective (Lehman 1974, Lehman et al, 1997, Kemerer and Slaughter 1999, Girba et al. 2005). What is not so obvious from these previous studies is how inheritance hierarchies in OO systems evolve in conjunction with, and as opposed to, that of system evolution. Moroever, inheritance is a form of coupling (Briand et al. 1999b). Anecdotal claims exist to suggest that coupling through inheritance is more favorable to that of non-inheritance coupling. We would therefore expect inheritance to be an alternative to non-inheritance coupling.

A vast majority of coupling and cohesion metrics abound in the literature relies on structural information, which captures relations, such as method calls or attributes usages. These metrics have been proved useful in different tasks, such as, assessment of design quality [1][2], impact analysis [3][4][5], prediction of software quality [6], and faults [7][8][9] identification of design patterns [10] etc. However, these structural metrics lack the ability to identify conceptual links, which, for example, specify implicit relationships encoded in identifiers and comments in source code.

The Object-Oriented (OO) software development technology was initially introduced in the early 1990s. Since then the OO paradigm has dominated mainstream software development (both in academia and industry) with languages such as C++ and Java. OO technology employs Classes together with Objects and their interdependencies to design and implement systems. OO introduced various underpinning approaches to software development which distinguish OO from traditional software development paradigm.

Inheritance is a cornerstone of OO paradigm. It is used to encapsulate a set of closely related functionality in a structured hierarchy where common functionality is added in one class and more specialized functionality of that class is added in other classes. The specialized classes inherit the common functionality from their super class and add their own extra functionality. The primary concern of inheritance is to promote reusability in a system. The overriding merits of reusability in software development are to: (i) remove the burden of re-writing an existing segment and (ii) to ease extensibility in a system. Furthermore, inheritance provides the facility for polymorphism.

In Java, a class can only inherit functionality from one other class, in C++ however, a class can inherit functionality from multiple classes in a system. To

facilitate multiple inheritances Java introduced the notion of interfaces. In this paper we want to include object oriented elements and basis on those elements we deduce some comparisons.

The remaining of this paper is organized as follows. We discuss problem domain in Section 2. In Section 3 we discuss about object oriented paradigm. In section 4 we discuss about Evolution and Recent Scenario. In section 5 we discuss about proposed methods. The conclusions and future directions are given in Section 6. Finally references are given.

2. Problem Domain

The current research on modeling and measuring the relationships among software components through coupling analysis is insufficient. Coupling measures are incomplete in their precision of definition and quantitative computation. In particular, current coupling measures do not reject the differences in and the connections between design level relationships and implementation level connections. Hence, the way coupling is used to solve problems is not satisfactory.

The basic theoretical results in this research, coupling-based analysis, have been applied to three specific engineering problems. Using the theory to solve three well studied problems demonstrates the power of this theory. Subsequent chapters describe, in detail, how the theory is applied to the problems. We introduce the concepts of the three sample problems: the class integration and test order problem, change impact analysis, and design pattern extraction using coupling measures. Figure 1 gives an overview of coupling-based analysis research and its applications. Coupling-based source code analysis (CBASCA) starts with parsing and analyzing the program source. Next, coupling measures are computed. Finally, the coupling measures are applied to class integration and test order, change impact analysis, and design pattern detection. The measures can also be applied to other software engineering problems.

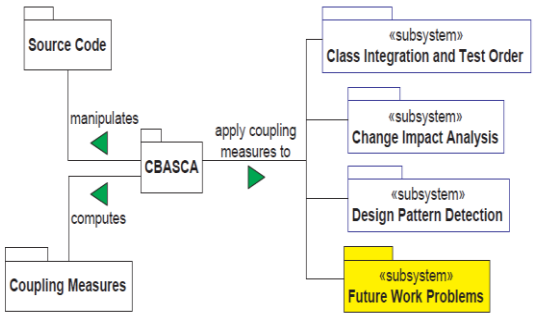

Figure 1 Application of Coupling Model

3. Quality Modules

The need of quality module is for better performance and output. So it is necessary that the produce module is good. A modeling notation consists of representations of the elements from which the software system will be built. An important aspect of the notation expresses the units of modularity, which, in an object-oriented notation, are classes. When we focus our models on classes, which are software modules, the resulting software is likely to be modular. However, this focus on software modules is no guarantee of a successful software system. The modules must be well designed. Good module design requires that the modules be Cohesive, Loosely coupled, Encapsulated and Reusable.

For a module to be cohesive, its functionality must be well defined and well focused. A class must have a clear, easily expressed objective. Cohesion refers to the degree to which the internal elements of a module are bound to or related to each other. For example, a Book class is likely to be a cohesive module because the concept of a book is clearly delineated in the minds of most people. An additional requirement for cohesiveness in an object-oriented system is that each class contains methods that relate only to that class. For example, the Book class should contain attributes and methods associated with a single book. Therefore, the Book class should not contain a method to list all books alphabetically or an attribute that represents the total number of books in the library.

Encapsulated modules are those that engage in data hiding. The attributes of an object should not be directly available to other objects. The attributes of the object are available only through a predefined interface that consists of the object's public methods. A user of a class, therefore, must only understand how to use the methods of the class in order to successfully interact with objects of that class. Encapsulation implies that knowledge of the implementation of a particular class is not necessary for users of the class so that users of a class are insulated from the details of the inner workings of the class. Modules that are not encapsulated allow users direct access to their attributes or require understanding of how the class is implemented in order to use it successfully. Software module reuse is a strategic objective of software engineering and is viewed as an essential approach for improving software development productivity. Reuse, therefore, should be a driving motivation in module design. The implication of designing for reuse is that class functionality should be as broad and general as possible. To achieve maximum reuse, a class may

require a few extra attributes or methods beyond the immediate scope of the target system.

4. Evolution Recent Scenario

In 2009, Yuming Zhou et al. [11] describe about the OO metrics that are investigated include cohesion, coupling, and inheritance metrics. Our results, based on Eclipse, indicate that: 1) The confounding effect of class size on the associations between OO metrics and change-proneness, in general, exists, regardless of whichever size metric is used; 2) the confounding effect of class size generally leads to an overestimate of the associations between OO metrics and change-proneness; and 3) for many OO metrics, the confounding effect of class size completely accounts for their associations with change-proneness or results in a change of the direction of the associations. These results strongly suggest that studies validating OO metrics on change-proneness should also consider class size as a confounding variable.

In 2010, V. Krishnapriya, et al. [12] proposed about the measurement to measure coupling between object (CBO), number of associations between classes (NASSocC), number of dependencies in metric (NDepIN) and number of dependencies out metric (NDepOut) in object oriented programming. A measurement is done for UML class diagrams and interface diagrams. The metric values of class and inheritance diagrams have been compared to prove which concept is good to use and beneficial for developers.

In 2007 , Maria Siniaalto et al. [13] reports the results from a comparative case study of three software development projects where the effect of TDD on program design was measured using objectoriented metrics. The results show that the effect of TDD on program design was not as evident as expected, but the test coverage was significantly superior to iterative test-last development.

In 2010, Simon Allier et al.[14] express existing definitions of coupling metrics using call graphs. We then compare the results of four different call graph construction algorithms with standard tool implementations of these metrics in an empirical study. Our results show important variations in coupling between standard and call graph-based calculations due to the support of dynamic features.

In 2010, Hongyu Pei Breivold et al.[15] primary studies for this review were identified based on a pre-defined search strategy and a multi-step selection process. Based on their research topics, we have identified four main categories of themes: software trends and patterns, evolution process support, evolvability characteristics addressed in OSS evolution, and examining OSS at software architecture level. A comprehensive overview and synthesis of these categories and related studies is presented as well.

In 2010, V. Krishnapriya et al. [16]presents a measurement to measure coupling between object (CBO), number of associations between classes (NASSocC), number of dependencies in metric (NDepIN) and number of dependencies out metric (NDepOut) in object oriented programming. A measurement is done for UML class diagrams and interface diagrams. The metric values of class and inheritance diagrams have been compared to prove which concept is good to use and beneficial for developers.

5. Proposed Method

In this paper we propose a novel java based computation and comparison method (JBCCM). In this method we taking three type of object oriented files for showing the computation. Those three files belong to C++, Java and C#. We first compute class, Inheritance, Interface, object and Line of Codes (LOC). Then we assume three databases based on several properties of C++, java, C#. Then we compare three files Based on class (BOC), Based on Inheritance (BOI), Based on Interfaces (BOIN), Based on Object (BOO) and Based on LOC (BOL).

Our proposed method consist of
Phase I- In first phase we have the choice of selecting file from three different object oriented programming. In our example we consider 3 different object oriented programs, which is c++, java and C#.[Figure 2]

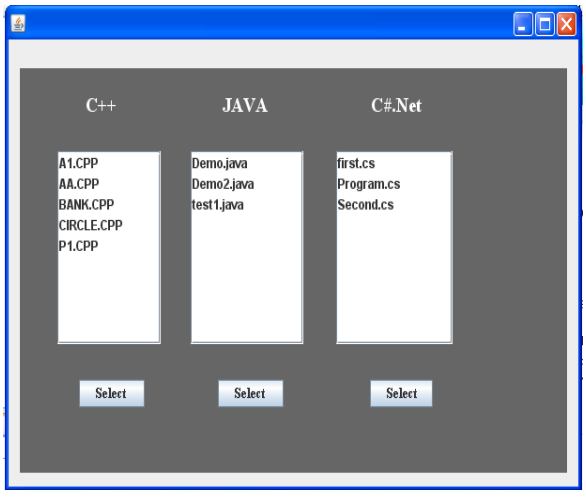

Figure 2: Selection from different Object Oriented Programs.

Phase II- In the next phase we select a file for example A1.cpp. Then the next window appears where we deduce the properties of the program like number of classes, Inheritance, Object, Polymorphism, Line of Code (LOC) etc. Based on the value we can perform some aggregation and association. In which changes can be done according to the properties of Object Oriented.

Phase III- In this phase we go to the comparison phase, where we can compare based on the different program methodology like c++, Java and C#. In this pahse we can evaluate our programming language according to the modularity of the program. [Figure 3]

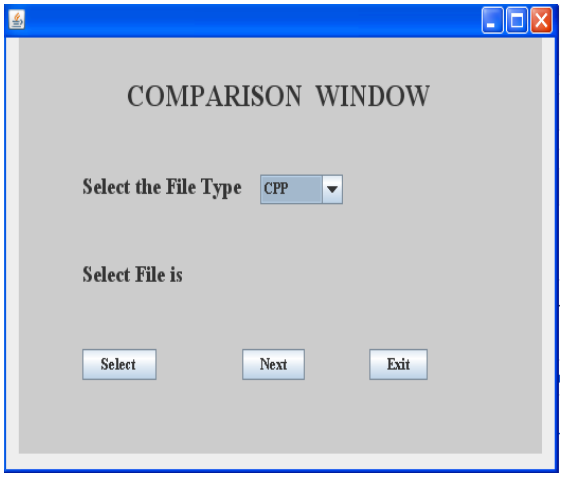

Figure 3: Comparison Window

Phase IV:- In this phase we compare the program and produce the result based on class(BOC),based on Inheritance(BOI), Based on Polymorphism(BOP),Based on Object(BOO). So we have four comparative parameter based on which we can perform the result evaluation.[Figure 4]

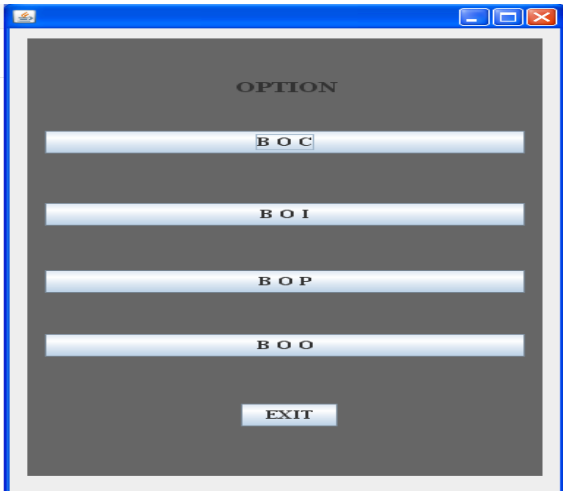

Figure 4: Option Window

Phase V:- In this phase we obtained the result and according to the result value we can prove that in which platform the algorithm is best. [Figure 5]

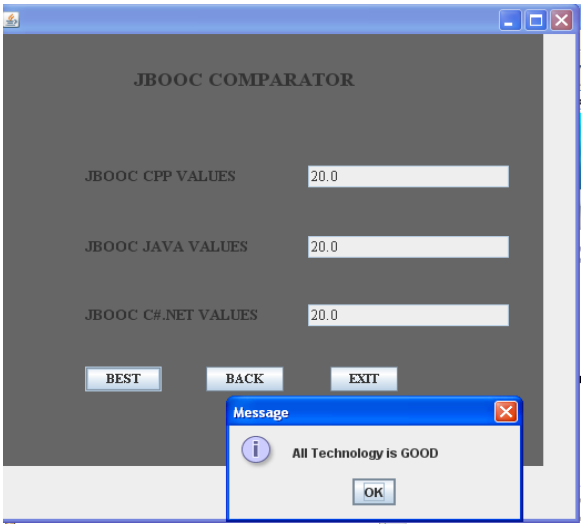

Figure 5: JBOOC Comparator

6. Conclusion

Classes in object-oriented systems, written in different programming languages, contain identifiers and comments which reflect concepts from the domain of the software system. This information can be used to measure the cohesion of software. To extract this information for cohesion measurement, Latent Semantic Indexing can be used in a manner similar to measuring the coherence of natural language texts.

In this paper we propose a novel java based computation and comparison method (JBCCM). In this method we taking three type of object oriented files for showing the computation. Those three files belong to C++, Java and C#. We first compute class, Inheritance, Interface, object and Line of Codes (LOC).

References

[1] Bansiya, J. and Davis, C. G., "A hierarchical model for object-oriented design quality assessment", IEEE TSE, vol. 28, no. 1, pp. 4-17., 2002.

[2] Briand, L. C., Wüst, J., Daly, J. W., and Porter, V. D., "Exploring the relationship between design measures and software quality in object-oriented systems", Journal of System and Software, vol. 51, no. 3, pp. 245-273., 2000.

[3] Briand, L., Wust, J., and Louinis, H., "Using Coupling Measurement for Impact Analysis in OO Systems", in IEEE ICSM'99, pp. 475-482., 1999.

[4] Poshyvanyk, D., Marcus, A., Ferenc, R., and Gyimóthy, T., "Using Information Retrieval based Coupling Measures for Impact Analysis", Empirical Software Engineering, vol. 14, no. 1, pp. 5-32., 2009.
[5] Wilkie, F. G. and Kitchenham, B. A., "Coupling measures and change ripples in C++ application software", JSS, vol. 52, pp. 157-164., 2000.
[6] Lawrie, D., Feild, H., and Binkley, D., "Leveraged Quality Assessment Using Information Retrieval Techniques", in ICPC'06, pp. 149-158., 2006.
[7] El-Emam, K. and Melo, K., "The Prediction of Faulty Classes Using Object-Oriented Design Metrics", NRC/ERB-1064, vol. NRC 43609, 1999.
[8] Gyimóthy, T., Ferenc, R., and Siket, I., "Empirical validation of OO metrics on open source software for fault prediction", TSE,vol.31/10,Oct' 2005.
[9] Quah, T.-S. and Thwin, M. M. T., "Application of neural networks for software quality prediction using OO metrics", in ICSM'03,pp. 116-125.
[10] Antoniol, G., Fiutem, R., and Cristoforetti, L., "Using Metrics to Identify Design Patterns in Object-Oriented Software", in Proc. of 5th IEEE METRICS'98, Bethesda, MD, pp. 23 - 34, 1998.
[11] Yuming Zhou, Hareton Leung and Baowen Xu, IEEE Transactions, September/October 2009.
[12] V. Krishnapriya and Dr. K. Ramar, 2010 International Conference on Advances in Computer Engineering,IEEE.
[13] Maria Siniaalto and Pekka Abrahamsson, First International Symposium on Empirical Software Engineering and Measurement,IEEE 2007.
[14] Simon Allier, St´ephane Vaucher, Bruno Dufour, and Houari Sahraoui, 2010 Working Conference on Source Code Analysis and Manipulation,IEEE.
[15] Hongyu Pei Breivold, Muhammad Aufeef Chauhan and Muhammad Ali Babar, 2010 Asia Pacific Software Engineering Conference,IEEE.
[16] V. Krishnapriya and Dr. K. Ramar," Exploring the Difference between Object Oriented Class Inheritance and Interfaces Using Coupling Measures", 2010, IEEE.