IJCSMS International Journal of Computer Science and Management Studies, Vol. 11, Issue 02, Aug 2011
ISSN (Online): 2231-5268
www.ijcsms.com


# Fault Prediction in Object Oriented System Using the Coupling and Cohesion of Classes

**Mr. Amol S. Dange[1], Prof. Dr. S. D. Joshi[2]**

[1]**Bharti Vidyapeeth Deemed University, Pune, Maharashtra, 411043, India**
***amol.dange@gmail.com***

[2]**Bharti Vidyapeeth Deemed University, Pune, Maharashtra, 411043, India**
***sdj@live.in***

**Abstract**

Building efficient systems is one of the main challenges for software developers, who have been concerned with dependability-related issues as they built and deployed. Lots of changes often needs including the nature of faults and failures and the complexity of systems. Sometimes accepting minor errors always need efforts to eliminate faults that might cause them is in the core of dependability. To this end various fault tolerance mechanisms have been investigated by researchers and used in industry. Unfortunately, more often than not these solutions exclusively focus on the implementation, ignoring other development phases, most importantly the earlier ones. This creates a dangerous gap between the requirement to build dependable (and fault prediction) systems and the fact that it is not dealt with until the implementation step.

A current software engineering gives attention towards only normal behavior with assumption that all faults can be removed during development. In fact every phase SDLC needs to be focused with phase-specific fault detection means.

We mean to conclude that SDLC requires:

- Integration of fault detection starting from requirement and architecture.
- Making fault detection-related decisions at each phase by explicit modeling of faults.
- Developing dedicated tools for fault detection modeling; providing domain-specific application-level fault prediction mechanisms.

Part I: Fault Prediction engineering: from requirements to code

Part II: Languages and Tools for engineering fault prediction systems

***Keywords:*** *– Design pattern, software metrics, measure theory, coupling, cohesion*

## I. INTRODUCTION

Trying to control software quality - and all related at tributes, it is obviously necessary to measure to what extend these attributes is achieved by a certain project. In this spirit, many software metrics have been established in the past.

In structured design and programming the importance of coupling and cohesion as main attributes related to the goodness of decomposition has been well understood; software engineering experts assure that designs with low coupling and high cohesion lead to projects that are both, more reliable and more maintainable.

The following list introduces the different types of coupling:

1. Data Coupling (communication via scalar parameters)
2. Stamp-Coupling (dependency induced by the type of structured parameters)
3. Control Coupling (parameters are used to control the behavior of a module)
4. Common Coupling (communication via shared global data)
5. Content Coupling (one module shares and/or changes the definition of another nodule)

For object oriented software, the coupling has not been considered with similar priorities. There are two main reasons for this negligence:

1. In structured design, there were few semantic guidelines to decompose a system into smaller subsystem. Consequently, syntactic aspects like size, coupling etc. played a major role. In contrast, in the object-oriented para digm, the main criterion for systems decomposition is the mapping of objects of the problem domain into classes or subsystems in the analysis/design model, thus reducing the relative importance of syntactic criteria.
2. Object-oriented analysis and design strive to incorporate data and related functionality into objects. This strategy in itself certainly reduces coupling between objects.

Therefore, explicitly controlling coupling does not seem to be as important as in structured (especially top-down) design.

However, since employing object-oriented mechanisms in it-self does not guarantee to really achieve minimum coupling. There is good reason to study coupling in object-oriented systems:

1. In many cases, data or operations cannot be unambiguously assigned to one or another class on the grounds of semantic aspects, thus designers does need some kind of additional criteria for such assignments.
2. Although introduction of classes as a powerful means for data abstraction reduces the data flow between ab straction units and therefore reduces also total coupling within a system, the number of variants of interdepen dency rises in comparison to conventional systems.
3. The principles of encapsulation and data abstraction, although fundamental to object-orientation, may be vio lated to different extents via the underlying programming language. This leads to different strength of de-facto coupling which should be taken into account.

Thus, coupling seems to be even more important in object-oriented systems:

- Coupling of client objects to a server object may intro duce change dependencies. The tighter the coupling, the harder the effects on the clients whenever a crucial as pect of the server is being changed.
- High coupling between two objects makes it harder to understand one of them in isolation. In contrast, low coupling leads to self-contained and thus easy to understand, maintainable objects.
- High coupling also increases the probability of remote effects, where errors in one object cause erroneous be havior of other objects. Again, lose coupling makes it easier to track down a certain error, which in turn im proves testability and eases debugging.

In this paper, based on a general notion of coupling, we at tempt to give appropriate definitions for coupling and cohe sion in object-oriented systems and identify a collection of dimensions that should be taken into account upon measuring these attributes. Analyzing the effects of coupling, it turns out that these can naturally be partitioned into two classes attributed to two different variants of coupling, namely Object coupling, Class coupling, and method level coupling respectively. Although our primary focus is on coupling as one of the most important internal attributes of software project, we must necessarily consider also cohesion because of the dual nature of these two attributes: Attempting to optimize a design with respect to coupling between abstractions (modules, classes, subsystems...) alone would trivially yield to a single giant abstraction with no coupling at the given level of ab straction. However, such an extreme solution can be avoided by considering also the antagonistic attribute cohesion (which would yield inadmissibly low values in the single-abstraction case).

## II. PRELIMINARIES

In this section we provide some prerequisites used throughout the rest of this paper Definition 1 clarifies some object- oriented parlance, while the following definitions are sup posed to give a preliminary idea of coupling in object oriented systems. These definitions will be refined in Section B.

**Definition 1 (Object oriented concepts):**

We will use the terms object and class according to the usual object-oriented terminology: A class provides the definition of structure (in-stance variables) and behavior (methods) of similar kinds of entities, an object is an instance of its respective class. Classes may be organized in inheritance hierarchies as super- and sub-classes.

**Definition2:**

Object coupling (OC) represents the coupling (in the sense of Definition2) resulting from state dependencies between objects during the run-time of a system.

**Definition3:**

Class coupling (CC) represents the coupling resulting from implementation dependencies in a system.

## III. COUPLING

Chidamber and Kemerer also define RFC (Response for a Class) as the union of the protocol a class offers to its clients and the protocols it requests from other classes. Measuring the total communication potential, this measure is obviously related to coupling and is not independent of CBO.

**Strength 1:**
Accessing the interface of any server class SC, provided SC is a stable class or features at least a stable interface, the most harmless type of Class coupling occurs, as no change dependencies are introduced.

**Strength 2:**
Changing the interface of an SC method called via an object local to one of CC's methods, only this latter method needs to be changed correspondingly. The same argument applies to the case where SC is the type of a parameter of a CC method.

**Strength 3:**
Changing the interface of an SC method in voked via a message sent to one of CC's instance variables of class SC, due to the class scope of instance variables, potentially all methods of CC are affected. This is why this case is less favorable than the above.

Similarly, changing the interface of a method of the super class SC of CC affects all methods of CC calling this super- class method. Thus, again potentially all methods of CC may be affected.
As a global variable is accessible from all methods of a class, the same argument applies for global variables, too.
Strengths 4 and 5: Following the same arguments as for strengths 2 and 3 and noticing that change dependencies are generally stronger when breaching the information hiding principle, these assignments result.

## IV. COHESION

Cohesion is an important attribute corresponding to the quality of the abstraction captured by the class under consideration. Good abstractions typically exhibit high cohesion. The original object-oriented cohesion metric as given by Chidamber and Kemerer (and clarified by the same au thors) represents an inverse measure for cohesion. They define Lack of Cohesion in Methods (LCOM) as the number of pairs of methods operating on disjoint sets of instance variables, reduced by the number of method pairs acting on at least one shared instance variable[6]. The definition given is reproduced below:
"Consider a Class $C_1$ with n methods $M_1$, $M_2$,,$M_n$. Let {Ij} = set of instance variables used by Method Mj.
There are n such sets $\{I_1\}$... $\{I_n\}$
Let $P = \{(I_i, I_j) \mid I_i \cap I_j = \emptyset\}$ and
$Q = \{(I_i, I_j) \mid I_i \cap I_j \neq \emptyset\}$. If all n sets $\{I_1\}$... $\{I_n\}$ are $\emptyset$ then let $P = \emptyset$.
$LCOM = |P| - |Q|$.

If |P| > |Q| = 0 otherwise.

So, LCOM is 2 - 1 = 1

Although the principle idea behind this definition seems very sensible, the resulting cohesion metric exhibits several anomalies with respect to the intuitive understanding of the attribute, the most important of which will be explained be low.

**The LCOM Metric: Lack of Cohesion in Methods**

The Lack of Cohesion in Methods metric calculations:

**LCOM1:**

Take each pair of methods in the class and determine the set of fields they each access. If they have disjointed sets of field accesses, the count R increases by one. If they share at least one field access, S increases by one. After considering each pair of methods:

RESULT = (R > S) ? (R - S) : 0

A low value indicates high coupling between methods. This also indicates potentially high reusability and good class design.

**LCOM2:**

This is an improved version of LCOM1. Say you define the following items in a class:

```
me:          Number of methods in a class
ac:          Number of attributes in a class
meA:         Number of methods that access the
             attribute a
sum(meA):    Sum of all meA over all the
             attributes in the class
mPr:         Number of private methods in a
             class
mPub:        Number of public methods in a
class mPro:  Number of protected methods in
             class
mPr+mPro):   sum of all (mPr+mPro) over all the
             attributes in the class
```

LCOM2 = 1- sum(meA)/(me*ac)

If the number of methods or variables in a class is zero (0), LCOM2 is undefined as displayed as zero.

**LCOM3:**

This is another improvement on LCOM1 and LCOM2 It is defined as follows:

LCOM3 = (me - sum(meA)/ac) / (me-1) where me, ac, meA, sum(meA) are as defined in LCM2. The following points should be noted about LCM3:

- The LCOM3 value varies between 0 and 2. LCOM3>1 indicates lack of cohesion and is considered a kind of alarm.
- If there is only one method in a class, LCOM 3 is undefined and also if there are no attributes in a class LCOM3 is also undefined and displayed as zero (0). Each of these different measures of LCOM has a unique way to calculate the value of LCOM.
- An extreme lack of cohesion such as LCOM3>1 indicates that the particular class should be split into two or more classes.
- If all the member attributes of a class are only accessed outside of the class and never accessed within the class, LCOM3 will show a high-value.
- A slightly high value of LCOM means that you can improve the design by either splitting the classes or re-arranging certain methods within a set of classes.

**LCOM5:** This is another improvement on LCOM,LCOM2 and LCOM3 It is defined as follows:

LCOM4 = (me – [sum(meA) –sum (mPr+mPro)]/ac) / (me-1)

where me, ac, meA, sum(meA),mPr, mPub, mPro are as defined in LCOM2.

## V. CONCLUSIONS AND FUTURE WORK

Having introduced a framework for a comprehensive metric for coupling in object-oriented systems on both, object and class levels, we were able to identify a basic ordinal metric for the contribution certain elementary constructs provide to coupling.

As an application of the framework, consider the trade-off discussed in this paper, namely, if using a (non-native) object is preferable to containing an object. Denoting the class of such an object by X, we find from LCOM5 of our framework that if X is stable, accessing an instance variable of this type X yields coupling strength 1 for the containing case.

Several open problems remain to be solved:

To achieve consistent and satisfying results, empirical data obtained from real-life software engineering projects need be analyzed with respect to the influence of the metrics proposed on external product attributes. This applies as well to the cohesion measures presented.

## ACKNOWLEDGMENT

The authors would like to thank Dr.S.D.Joshi for several fruitful discussions.

## REFERENCES

[1] L.C. Briand, S. Morasca, and V.R. Basili, “Property-Based Software Engineering Measurements,” IEEE Trans. Software Eng., vol. 22, no. 1, pp. 68-85, Jan. 1996.

[2] G. Antoniol, G. Canfora, G. Casazza, and A. De Lucia, “Identifying the Starting Impact Set of a Maintenance and Reengineering”, Proc. Fourth European Conf. Software Maintenance, pp. 227-230.

[3] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, “Recovering Traceability Links between Code and Documentation,” IEEE

Trans. Software Eng., vol. 28, no. 10, pp. 970-983, Oct. 2002.

[4] E. Arisholm, L.C. Briand, and A. Foyen, "Dynamic Coupling Measurement for Object-Oriented Software," IEEE Trans. Software Eng., vol. 30, no. 8, pp. 491-506, Aug. 2004.

[5] 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.

[6] V.R. Basili, L.C. Briand, and W.L. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators," IEEE Trans. Software Eng., vol. 22, no. 10, pp. 751-761, Oct. 1996.

[7]M.W. Berry, "Large Scale Singular ValueComputations," Int'lJ Supercomputer Applications, vol. 6, pp. 13-49, 1992

[8] J. Bieman and B.K. Kang, "Cohesion and Reuse in an Object- Oriented System," Proc. Symp. Software Reusability, pp. 259-262, Apr. 1995.

[9] L. Briand, W. Melo, and J. Wust, "Assessing the Applicability of Fault-Proneness Models Across Object-Oriented Software Projects," IEEE Trans. Software Eng., vol. 28, no. 7, pp. 706-720, July 2002.

[10] L.C. Briand, J.W. Daly, V. Porter, and J. Wu¨ st, "A Comprehensive Empirical Validation of Design Measures for Object-Oriented Systems," Proc. Fifth IEEE Int'l Software Metrics Symp, pp. 43-53, Nov. 1998

[11] L.C. Briand, J.W. Daly, and J. Wu¨ st, "A Unified Framework for Cohesion Measurement in Object-Oriented Systems," Empirical Software Eng., vol. 3, no. 1, pp. 65-117, 1998.