Investigation of Metrics for Object-Oriented Design Logical Stability

Mahmoud O. Elish
Department of Computer Science
George Mason University
Fairfax, VA 22030-4400, USA
melish@gmu.edu

David Rine
Department of Computer Science
George Mason University
Fairfax, VA 22030-4400, USA
drine@gmu.edu

Abstract

As changes are made to an object-oriented design, its structure and/or behavior may be affected. Modifications made to one class can have ripple effects on other classes in the design. The stability of an object-oriented design indicates its resistance to interclass propagation of changes that the design would have when it is modified. There are two aspects of design stability: logical stability and performance stability. Logical stability is concerned with design structure, whereas performance stability is concerned with design behavior. In this study, the object-oriented design metrics proposed by Chidamber and Kemerer were adopted as candidate indicators of the logical stability of object-oriented designs. The objective was to investigate whether or not there are correlations between these metrics and the logical stability of classes. The experimental results indicated that WMC, DIT, CBO, RFC, and LCOM metrics are negatively correlated with the logical stability of classes. However, no correlation was found between NOC metric and the logical stability of classes.

Keywords *– Design stability, maintainability, metrics, object-oriented designs.*

1. Introduction

Software maintenance is inevitable if software systems need to remain useful in their environments. Changes are necessary to continue increasing the value of software. Major portion of software maintenance activities is devoted to the modification of the software [16]. Due to the coupling between software artifacts, changes made to one artifact can ripple throughout the software requiring further changes to other artifacts.

The ISO/IEC 9126 software quality standard [17] defines six quality attributes of software: functionality, reliability, efficiency, usability, portability, and maintainability. According to this standard, maintainability has four quality attributes: analyzability, changeability, testability, and stability. Stability is defined as those attributes of software that bear on the risk of unexpected effect of modifications.

According to DeMarco's principle [4]: "You cannot control what you cannot measure." Some stability measures were proposed for procedural programs and designs [1, 2, 8, 11, 13, 14, 15, 16], and one for object-oriented design [12]. There are some weaknesses in the existing measures that prevent their wide acceptance. These problems are identified in Section 3.

In this study, the object-oriented design metrics proposed by Chidamber and Kemerer [3] were adopted as candidate indicators of the logical stability of object-oriented designs. These metrics were selected because they are well defined, widely used, and were shown to be valid maintainability predictors. The objective of this study was to investigate whether or not there are correlations between these metrics and the logical stability of classes.

This paper is organized as follows. Section 2 gives some technical backgrounds. Section 3 reviews related work. Section 4 discusses the experiment and its results. Section 5 concludes the paper and gives directions for future work.

2. Background

This section gives brief background about the stability concept of object-oriented designs, types of class-level changes, and the metrics defined by Chidamber and Kemerer.

2.1. Stability of Object-Oriented Designs

As changes are made to an object-oriented design, its structure and/or behavior may be affected. Modifications made to one class can have ripple effects on other classes in the design. Ripple effects may or may not be desirable (do not require additional changes). As an example of desirable ripple effect, consider a concrete superclass with a method that sort array of numbers using selection sort algorithm. Assume that this method is inherited by many subclasses. Now consider changing the implementation of this method by replacing the selection sort algorithm with more efficient algorithm such as merge sort. Such modification will have desirable ripple effect on all subclasses that inherit this method, and no additional changes are required to be made to these subclasses.

As an example of undesirable ripple effect, consider an interface class with a set of method signatures. These methods are implemented by the concrete classes that implement this interface class. Now consider changing the interface class by adding, deleting, or modifying a method signature. Such modification will impact all classes that implement the interface class, and will require additional changes to be made to these classes.

A good object-oriented design from stability standpoint should localize changes as much as possible to classes on which alterations are made. The stability of an object-oriented design indicates its resistance to interclass propagation of changes that the design would have when it is modified. Class stability is the likelihood that the class will not be change-prone as a consequence of changes made to other classes in the design.

There are two aspects of design stability: logical stability that is concerned with the stability of design structure, and performance stability that is concerned with the stability of design behavior. This paper focuses on the logical stability of object-oriented designs.

2.2. Types of Class-level Changes

Change impact analysis literature has identified possible types of changes that can be made to object-oriented designs/software, for example, Kung et al. [9], and Li and Offutt [10]. These types of changes can be classified into two main categories: design-level changes, and class-level changes. Examples of design-level changes include adding a class, deleting a class, adding an association between two classes, and restructuring the class hierarchy.

At the class-level, various types of changes can be applied to the attributes, methods, and assertions of classes. Attribute changes include addition, deletion, data type change, value change, and scope change. Method changes include addition, deletion, signature change, implementation change, and scope change. Assertion changes include modifications to class invariants, and method preconditions and postconditions.

Changes may have syntactic and/or semantic impact. The syntactic impacts cause compilation errors and can be determined through static analysis. The semantic impacts, however, do not cause compilation errors and can be determined during run time. Performance stability shall address changes with semantic impact. Since the focus of this research is on the logical stability of object-oriented designs in terms of interclass propagation of changes, it is limited to those class-level changes that have syntactic impact. Table 1 lists all possible types of class-level changes with syntactic impact. It also identifies the set of classes that are syntactically impacted as a result of each type of change.

To further illustrate Table 1, consider the simple class diagram shown in Figure 1. The diagram consists of three classes C1, C2, and C3. C2 uses some of the attributes and methods of C1, and C3 inherits the attributes and methods of C1. As one example, assume that attribute B of C1 is referenced by method Y of C1, method M of C2 and method Z of C3. Consider changing the scope of B from public to protected. This change will syntactically impact C2.

As another example, assume that method X of C1 is invoked by method Y of C1 and method N of C2. Consider deleting X. This will syntactically impact C1 and C2.

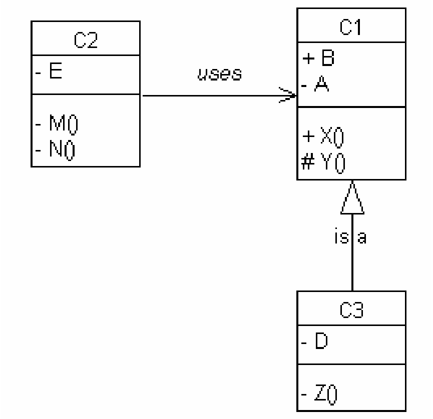

Figure1. Simple class diagram example

2.3. Metrics by Chidamber & Kemerer

Chidamber and Kemerer [3] proposed a metrics suite for object-oriented design that consists of the following metrics:

- *Weighted Methods per Class (WMC)* – It is defined as the sum of the complexities of all methods of a class.
- *Depth of Inheritance Tree (DIT)* – It is defined as the maximum length from a class to the root class in the inheritance tree.
- *Number of Children (NOC)* – It is defined as the number of immediate subclasses of a class.

Table 1. Possible class-level changes with syntactic impact

	Change Type	Syntactically Impacted Classes
Attribute (*A*) of Class (*C*)	Data type	*X*
	Delete	*X*
	Scope (public to private)	$X - \{C\}$
	Scope (protected to private)	$X - \{C\}$
	Scope (public to protected)	$X - [Z \cup \{C\}]$
Method (*M*) of Class (*C*)	Return data type	*Y*
	Signature	*Y*
	Delete	*Y*
	Scope (public to private)	$Y - \{C\}$
	Scope (protected to private)	$Y - \{C\}$
	Scope (public to protected)	$Y - [Z \cup \{C\}]$
X is the set of classes, including C, that reference A		
Y is the set of classes, including C, that invoke M		
Z is the set of direct and indirect subclasses of C		

- *Coupling Between Object classes (CBO)* – It is defined as the number of classes to which a class is coupled. Two classes are coupled if one uses methods and/or instance variables of another.
- *Response For a Class (RFC)* – It is defined as number of methods in the set of all methods that can be invoked in response to a message sent to an object of a class.
- *Lack of COhesion in Methods (LCOM)* – It is defined as the number of different methods within a class that reference a given instance variable.

3. Related Work

The term ripple effect was introduced by Haney [8] to describe that a change in one module may necessitate a change in another module. Haney used a technique called module connection analysis that is based on applied matrix algebra to estimate the number of changes needed to stabilize a system. Soong [13] developed techniques based on the method of connectivity matrix to measure program information structures for their stability characteristics. Myers [11] used matrix to describe dependencies between system's modules, and then used it to predict the stability of the system. There are two major weaknesses in the stability measures proposed by Haney, Soong, and Myers. First, these measures were not validated because their inputs are difficult to obtain [14]. Second, they assume that all modifications to a module have the same ripple effect [14].

Yau and Collofello [14] developed algorithm to compute the logical stability of a program and its modules using McCabe's complexity measure. The computation is based on the connection of modules in a program by parameters and global variables. They then proposed algorithm for computing design stability [16]. The major problem with these algorithms is that they take too much computation time to be practicable for large programs [1, 15].

Yau and Chang [15] developed more efficient algorithm for computing ripple effect. This algorithm, however, treats modules as black boxes and does not consider information inside modules. Black [1, 2] proposed an approximation algorithm for Yau and Collofello's algorithm to compute ripple effect for C programs.

All the researches described so far were merely on measuring the stability of procedural programs and their designs. Samadzadeh and Khan [12] proposed a stability metric for object-oriented software systems based on the assumptions that different objects in a program make about one another as a result of parameter coupling. This metric ignores inheritance coupling resulting from is-a relationships between classes, and component coupling resulting from is-part-of relationships between classes.

As guidelines for building stable designs, Fayad [5, 6, 7] discussed the concepts of Enduring Business Themes (EBTs), Business Objects (BOs), and Industrial Objects (IOs), and how they can be used to build a stable design. EBTs are those core concepts of the system that remain stable over time. BOs are externally stable over time, but may have internal changes. IOs are peripheral and unstable objects of the system. Fayad argued that a design model based on these concepts would reduce reengineering, and thus yield a more stable design.

4. The Experiment

The objective of this experiment was to investigate whether the object-oriented design metrics proposed by Chidamber and Kemerer are good indicators of the logical stability of classes. In other words, the conduced experiment aimed to test for existence of significant correlations between these metrics and the logical stability of classes.

4.1. Hypotheses

Two Hypotheses were tested by this experiment:

- *Hypothesis 1* – WMC, DIT, CBO, RFC, and LCOM metrics are negatively correlated with the logical stability of classes.
- *Hypothesis 2* – There is no correlation between NOC metric and the logical stability of classes.

4.2. Subjects

The experimental subjects used in this study were three arbitrarily selected open source Java software: Jxplorer version 2.1.001, JE (Just an Editor) version 1.65, and Phex version 0.7.3. They vary in number of classes and application domain. The source codes of these systems can be freely downloaded from the web [18]. Although the subjects are source codes and not designs, only design metrics and information were extracted from them.

Jxplorer is a security application with 157 classes for browsing LDAP (Lightweight Directory Access Protocol). JE is a programming editor application with 252 classes. It provides syntax highlighting and can also be used as an editor panel in other Java applications or applets. Phex is a networking application with 303 classes that offers an automatic search functionality to find new download candidates and uses swarming to resume the downloads across different hosts. Table 2 gives various size metrics of these three systems.

4.3. Procedure

Three main steps were carried out in this experiment. In the first step, the object-oriented design metrics proposed by Chidamber and Kemerer were collected from the classes of each subject system using Jstyle 4.6 metrics tool. WMC of a class was calculated by summing the cyclomatic complexities of the methods defined in this class. DIT of a class was measured by determining the level of this class in the inheritance tree. NOC of a class was computed by counting the number of direct subclasses of this class. CBO of a class was measured by counting the number of classes that this class depends on and the number of classes that depend on this particular class. In this context, a dependency between two classes means that one uses methods and/or instance variables of another. Classes that are depended on by a class and also depend on it were counted once. RFC of a class was calculated by counting the number of methods, internal and external, available to this class. LCOM of a class was calculated by subtracting the number of method pairs whose similarity is not zero from the number of method pairs whose similarity is zero. The similarity of two methods is computed by counting the number of instance variables that are used by both of them.

In the second step, the logical stability of each class was calculated using the algorithm given in the appendix. To calculate the logical stability of a class (say *C*), the algorithm applies all possible class-level changes with syntactic impact (see Table 1) to the attributes and methods of all other classes in the design. Changes are applied one at a time, and they are all applied to the original design. For each change, the algorithm computes the set of syntactically impacted classes and determines whether or not class *C* is among them. The number of times that class *C* is found impacted divided by the total number of possible changes represents the likelihood that class *C* will be change-prone as a result of a class-level change made else where in the design. So the logical stability of class *C* is simply one minus that ratio. Given that types of changes are unpredictable and that no studies found that some changes are more likely than others, it was assumed that all types of class-level changes are equally likely.

In the third step, correlation analyses were performed at 0.05 level of significance (95% confidence level) to test for existence of correlation between each one of the six investigated metrics and the calculated logical stability of classes. For each case, correlation coefficient was computed to determine the significance of correlation.

4.4. Results

Table 3 provides descriptive statistics for Chidamber and Kemerer's metrics data that were collected from the classes of the subject systems. On the one hand, wide variations were found in WMC, CBO, RFC, LCOM metrics from one system to another. On the other hand, all of the three systems have a median value of zero for both DIT and NOC metrics. This indicates that the use of inheritance in these systems is limited.

Two results were obtained from the correlation analyses performed at 0.05 level of significance (95% confidence level) to test for existence of correlations between the investigated metrics and the logical stability of classes. First, there exists a negative correlation between each one of WMC, DIT, CBO, RFC, and LCOM metrics and the logical stability of classes. Second, there does not exist a correlation between NOC metric and the logical stability of classes. These results support the hypotheses.

Table 4 provides the computed correlation coefficients between each metric and the logical stability of classes for each system. Some observations can be obtained from this table. CBO and RFC metrics are strongly correlated with the logical stability of classes, with correlation coefficients of -0.69 and -0.84 on average respectively. WMC and LCOM metrics come next with reasonable correlation coefficients of -0.61 and -0.50 on average respectively. DIT metric is weakly correlated with the logical stability of classes, with a correlation coefficient of -0.21 on average. This may due to the low variations of DIT values in the subject systems because of their limited use of inheritance.

4.5. Discussion

The negative correlations between WMC, DIT, CBO, RFC, and LCOM metrics and the logical stability of classes can be explained as follows. High WMC of a class suggests that this class has many methods and/or its methods have high complexity. This may increase the likelihood of having some methods that use methods and/or instance variables of other classes. If so, this makes this class depends upon other classes, and thus reduces its stability.

The higher DIT of a class, the more ancestor classes it has. A subclass does not only depend upon its direct superclass, but also upon its ancestor classes as it inherits their features. This decreases the stability of subclasses because changes to their ancestors may necessitate changing them.

High CBO of a class means that this class depends upon many classes (outgoing dependencies) and/or many classes depend upon it (incoming dependencies). The outgoing dependencies of a class reduce its stability since they represent external influence to make it change.

The higher RFC of a class, the higher the number of internal and external methods available to this class. The external methods make this class depends upon the classes in which these methods are defined. This in turn may require modifying this class whenever these external methods are modified. So this decreases class stability.

High LCOM of a class suggests that this class is lowly cohesive and does not promote encapsulation. This may increase the likelihood of this class being dependant upon other classes. If so, this reduces the stability of this class.

The absence of correlation between NOC metric and the logical stability of classes can be explained as follows. The more subclasses a class has, the higher its incoming dependencies since these subclasses depend upon it. Since only the outgoing dependencies of a class affect its stability, NOC metric is not expected to have correlation with class stability.

Table 2. Size metrics for the subject systems

	Jxplorer	**JE**	**Phex**
No. of Classes	157	252	303
No. of Code Lines	30112	18445	36729
No. of Comment Lines	17015	3171	13837
No. of Blank Lines	10770	1608	7162
No. of Declarative Statements	6044	4221	7405
No. of Executable Statements	13266	10768	13472

Table 3. Descriptive statistics of the classes in the subject systems

		WMC	DIT	NOC	CBO	RFC	LCOM	Class Logical Stability
Jxplorer	**Average**	31.90	0.27	0.28	11.66	22.82	181.35	96.39%
	Median	16	0	0	7	14	28	97.45%
	Std. Dev.	40.03	0.51	1.39	14.02	26.32	425.75	4.34%
	Maximum	223	3	14	88	172	2701	100.00%
	Minimum	0	0	0	0	0	0	76.43%
JE	**Average**	17.13	0.48	0.45	7.24	13.98	110.11	98.51%
	Median	6	0	0	4	7	3	99.01%
	Std. Dev.	46.19	0.62	2.18	10.82	28.09	806.98	2.25%
	Maximum	550	3	29	103	334	11325	100.00%
	Minimum	0	0	0	0	0	0	71.83%
Phex	**Average**	14.67	0.40	0.27	10.26	15.39	58.37	98.37%
	Median	9	0	0	5	10	10	99.01%
	Std. Dev.	17.31	0.68	2.23	14.55	19.37	169.66	2.33%
	Maximum	133	2	34	125	136	2080	100.00%
	Minimum	0	0	0	0	0	0	84.82%

Table 4. Correlation coefficients

	WMC	DIT	NOC	CBO	RFC	LCOM
jxplorer	-0.51	-0.20	-0.05	-0.64	-0.77	-0.49
JE	-0.71	-0.27	0.03	-0.71	-0.88	-0.59
Phex	-0.60	-0.16	0.06	-0.73	-0.86	-0.41
Average	-0.61	-0.21	0.01	-0.69	-0.84	-0.50
Std. Dev.	0.10	0.05	0.06	0.04	0.06	0.09

5. Conclusions

Stability is one of the most desirable features of any software design. If the stability of a design is poor, the impact of any original modification on it is large, i.e. high amplification of changes throughout the design is expected. Consequently, the maintenance cost and effort may turn to be higher than what was estimated, and the software reliability may also suffer due to the introduction of possible new defects. Therefore, the availability of a verified set of logical stability metrics for object-oriented designs represents early crucial signals of any out-of-control situation that may occur during maintenance.

The goal of this study was to investigate whether Chidamber and Kemerer's metrics represent good indicators of the logical stability of classes. The experimental results concluded that WMC, DIT, CBO, RFC, and LCOM metrics are negatively correlated with the logical stability of classes. In addition, CBO and RFC metric were found to be good indicators of the logical stability of classes. However, no correlation was found between NOC metric and the logical stability of classes.

This study represents the first step toward a suite of logical stability metrics for object-oriented designs. Future works include investigation of more object-oriented design metrics for logical stability, as well as deriving new metrics. Once a verified set of these metrics is identified, various logical stability prediction models can be developed.

6. References

[1] S. Black, "Computing ripple effect for software maintenance," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 13, pp. 263-279, 2001.

[2] S. Black, “Measuring Ripple Effect for Software Maintenance,” *Proceedings of the International Conference on Software Maintenance*, Sep. 1999.

[3] S. Chidamber and C. Kemerer, “A Metrics Suite for Object Oriented Design,” *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476-493, June 1994.

[4] T. DeMarco, *Controlling Software Projects: Management, Measurement & Estimation*, Prentice-Hall, 1982.

[5] M. Fayad and A. Altman, “An Introduction to Software Stability,” *Communications of the ACM*, vol. 44, no. 9, pp. 95-98, Sep. 2001.

[6] M. Fayad, “Accomplishing Software Stability,” *Communications of the ACM*, vol. 45, no. 1, pp. 111-115, Jan. 2002.

[7] M. Fayad, “How to Deal with Software Stability,” *Communications of the ACM*, vol. 45, no. 4, pp. 109-112, Apr. 2002.

[8] F. Haney, “Module Connection Analysis – A Tool for Scheduling Software Debugging Activities,” *Proceedings of the AFIPS Fall Joint Computer Conference*, pp. 173-179, Dec. 1972.

[9] D. Kung, J. Gao, P. Hsia, F. Wen, Y. Toyoshima, and C. Chen, “Change Impact Identification in Object Oriented Software Maintenance,” *Proceedings of the International Conference on Software Maintenance*, pp. 202-211, 1994.

[10] L. Li and J. Offutt, “Algorithmic Analysis of the Impact of Changes to Object-Oriented Software,” *Proceedings of the International Conference on Software Maintenance*, pp. 171-184, 1996.

[11] G. Myers, *Reliable Software through Composite Design*, Petrocelli/Charter, pp. 137-149, 1975.

[12] M. Samadzadeh and S. Khan, “Stability, Coupling, and Cohesion of Object-Oriented Software Systems,” *Proceedings of the 22nd Annual ACM Computer Science Conference*, pp. 312-319, 1994.

[13] N. Soong, “A Program Stability Measure,” *Proceedings of the ACM Annual Conference*, pp. 163-173, 1977.

[14] S. Yau and J. Collofello, “Some Stability Measures for Software Maintenance,” *IEEE Transactions on Software Engineering*, vol. SE-6, no. 6, pp. 545-552, Nov. 1980.

[15] S. Yau and S. Chang, “Estimating Logical Stability in Software Maintenance,” *Proceedings of the 8th Annual International Computer Software and Applications Conference*, pp. 109-119, Nov. 1984.

[16] S. Yau and J. Collofello, “Design Stability Measures for Software Maintenance,” *IEEE Transactions on Software Engineering*, vol. SE-11, no. 9, pp. 849-856, Sep. 1985.

[17] ISO/IEC 9126: Information technology - Software Product Evaluation - Quality characteristics and guidelines for their use - 1991.

[18] SourceForge.net, Online at http://sourceforge.net/.

7. Appendix: Algorithm for calculating class logical stability

```
ClassLogicalStability (Class C, ObjectOrientedDesign OOD)
   Input:  Class C, Object-Oriented Design OOD in which class C exist
   Output: The likelihood that C will not be change-prone as a result of
           a class-level change with syntactic impact made to another class in OOD
   BEGIN
       TotalNumOfChanges = 0
       NumOfChangesImpactedClassC = 0
       FOR each class A except C in OOD DO
          FOR each attribute T in A DO
             FOR each type of attribute change I DO   /* see Table 1 */
                IF I can be applied to T THEN
                   IC = {set of classes impacted by applying I to T}
                   IF C ∈ IC THEN
                      NumOfChangesImpactedClassC = NumOfChangesImpactedClassC + 1
                   ENDIF
                   TotalNumOfChanges = TotalNumOfChanges + 1
                ENDIF
             ENDFOR
          ENDFOR
          FOR each method M in A DO
             FOR each type of method change J DO     /* see Table 1 */
                IF J can be applied to M THEN
                   IC = {set of classes impacted by applying J to M}
                   IF C ∈ IC THEN
                      NumOfChangesImpactedClassC = NumOfChangesImpactedClassC + 1
                   ENDIF
                   TotalNumOfChanges = TotalNumOfChanges + 1
                ENDIF
             ENDFOR
          ENDFOR
       ENDFOR
       RETURN (1 - (NumOfChangesImpactedClassC / TotalNumOfChanges))
   END
```