Empirical Investigation of the Relationship between Interaction Level Metric and Class Defects

Kuo Hua Chung, Chia Hung Kao, Hewijin Christine Jiau
Department of Electrical Engineering
National Cheng Kung University
{leomon,basara}@nature.ee.ncku.edu.tw jiauhjc@mail.ncku.edu.tw

Abstract

The object-oriented paradigm has become popular in recent years. In order to better evaluate and control the quality of object-oriented software systems, many object-oriented design metrics proposed to indicate the design properties that influence software quality during design phase. Design metrics proposed should be empirically validated to demonstrate the usefulness. The major focus of our work is to empirically investigate the relationship between class defect count and an existing design complexity metric, Interaction Level (IL). Several open source projects are studied to perform such an investigation. The result shows this metric can be a useful reliability indicator at the class level.

Keywords：Object-oriented design metrics, class defect, metrics validation.

1 Introduction

Software reliability is more emphasized on software market. Thus, software developer would like to build a reliable software product within limited time and budget. Software failures may occur due to many causes. In particular, during software design, the design decisions influence the software reliability. Therefore, in order to produce high quality software, a strong emphasis on design aspects, especially during the early design phase, is necessary since early correction actions are less costly. One way to accomplish this goal is design metrics. Software design metrics are defined to measure some particular design perspectives and provide developers a quantified approach to evaluate the design. Thus, software metrics can be used to set up a model to identify the part at high risk of software system and improve the reliability. Many traditional software product metrics, such as McCabe's cyclomatic complexity [11] or line of code [13] are proposed. It has been shown that these metrics are associated with defects and maintenance performance [8][12]. However, as object-oriented technology is popular, traditional software metrics cannot model the key concepts in object-oriented design, such as inheritance or encapsulation. Hence, many object-oriented metrics are developed. One of the object-oriented design metrics suites is CK metrics [8]. Many researches studied the relationship between software quality and CK metrics [5][14].

However, besides CK metrics, some metrics can capture the information that CK metrics cannot. The metric we focus on is Interaction Level (IL) proposed by Abbott et al. [3]. IL captures class interface and attribute design information which CK metrics ignore. In early design phase, CK metrics measure the structural relationship such as CBO (Coupling between Object Classes), and DIT (Depth of Inheritance Tree). CK metrics also include a complexity metric, WMC (Weighted Method per Class), usually defined as the number of methods of class in early design phase. IL exploits the class interface and attribute information to provide a more accurate complexity measurement than simple method counting. In order to demonstrate the usefulness of each design metric, empirical validation should be performed. In priori literature, the association between the interface information captured by IL and maintenance performance is validated by Bandi et al. [4]. Our focus is to empirically investigate the relationship between IL and class defect count.

The organization of the rest of this thesis is as following: In section 2, we introduce IL metric. In section 3, we present the model and experiment hypotheses. Section 4 describes the design of the experiments, data collection process and the experiment result and Section 5 gives the conclusion.

2 Definition of Interaction Level

IL provides an estimate design complexity metric when class interface and attributes are defined. It is calculated based on the maximum interactions between variables (attributes and parameters) within a method. There exist two variables A and B, if the value of state of B is (directly or indirectly) influenced by the value or state of A, there is an interaction from A to B. The definition of IL is as following:

IL = k1 * (value based on number of interactions) + k2 * (value based on strength of interactions)

Because the complexity of each interaction

Table 1: Size of different data types

Type	Size value
Boolean	1
Integer or Character	2
Real	3
Array	+2
Object	5

should be different, strength is the measurement to distinguish the complexity of interaction constituted from different data types. The strength of interaction is defined as the product of size of the variables involved in an interaction. The size values of different data type are specified based on [4] and we made some modifications. We explain such modification in next section. The size values are list in Table 1. It is necessary to use both number and strength because they typically have an inverse relationship that decrease in either number or strength could increase the other and vice versa. Here, the constant k1 and k2 means the different importance of number and strength of interaction. We set these constants to one for simplicity and balance the influence of number and strength of interaction. IL value can be aggregated to different level of granularity. IL for a class is the summation of all methods' IL value in the class . In our work, constructors and utility functions are excluded for computing IL metric value.

3 Models and Hypotheses

IL metric has been subjectively validated by comparing IL values and design experts preference [3] and empirically validated the relationship between IL and maintainability [4] by simple controlled experiments. As the previous sections describe, IL is a more appropriate design complexity metric and to be an indicator of class defect count. However, it lacks the empirical validation. Therefore, our main objective was to focus on exploring the relationship between the object-oriented design complexity metric and the defect count at the class level. After such relationship has been validated, IL can be used as the class defects indicator. The design of the software system can be evaluated and developer can figure out the classes at high risk. That is, more defects may occur within the class. Designer can try to redesign the part at high risk or use other approaches in order to raise the reliability. In order to validate the relationship, the following hypothesis should be statistically tested.

H1: A class has higher IL value will be associated with higher number of defects.

The dependent variable in our analysis is defect count for a class. In previous work, researchers proposed binary classification of defects data and used logistic regression models to measure the impact of design complexity on defects proneness [5][6]. The drawback of using such binary classification scheme is that a class with one defect cannot be distinguished from a class with ten defects. As a result, the true variance of defects in data sample may not be captured in the empirical analysis. Therefore, we use the actual defect count as the dependent variable in our analysis. From the above, the fist model is given below:

(1) $\boldsymbol{Defects = \beta 0 + \beta 1 \times IL}$

A class has high IL value may be caused by

- Large number of interaction. Large number of interaction represents the interactions of the data item within its methods are complex. That means there may be complex data dependency relationships within the class’ methods. Let’s take a simple example to illustrate this. In a method, two data item interact with each other under the situation that they appear in the same control flow path. Large number of interactions may imply the method consists of many control flow paths. More control flow path implies more complexity for developer. Therefore, when the complexity of class development for developer is high, it results in more defects.

- Large strength of interaction. The size of data type means the relative degree of complexity to correctly use a data item in the interaction. For example, using an object reference data item will be more complex than using a primitive data type in an interaction. That may cause different degree of influence on class reliability. Therefore, the size of object should be larger than primitive data type. Besides, the size of array type is set to “+2”. It means the size of an integer array is the size of integer plus two. The reason to make such setting is, when using array, developers need to pay attention to the size and index of this array. The strength of interaction means the complexity to correctly arrange the interaction to achieve the intended effect. Thus, large strength may result in more defects.

- Both.

Therefore, it is suggested that higher interaction level correlates with increased difficulty in determining how to develop or implement a design. That means a design with higher interaction level will result in the detailed design and the implementation of this class to be more difficult. There will be more defects in this class.

However, using IL to indicate the design complexity of an object is limited, because it only indicates the complexity for interaction between its surface and its interior of an object. The focus of IL is only the internal complexity of class' methods. Therefore, we try to find other existing metrics exploited to describe design complexity of different viewpoint. The metric selected is Coupling between Object Classes (CBO) metric of CK metrics suite [7], which models the coupling or structural relationship of an object to other objects in software system. Coupling means a class uses the methods or instance variables of other classes. CBO of a class is defined as the number of other classes in the system it is coupled. A class has high CBO value means lots of classes it depends. The meanings of CBO in software development are stated as the following:

- The higher the CBO value of class, the more rigorous the testing needs to be.
- In evaluating class reliability, higher CBO value leads to the more difficulties for designers and developers to manage and correctly use these classes provide services.

In some cases, the classes have similar IL values but different CBO values. In such situations, CBO is the key point addressing the different design complexity. For example, the complexity to design a class use ten objects of the same class should differ from design a class use ten objects of different types. CBO and IL should be complementary since they model different design perspectives. Then we have the second model.

(2) $$Defects = \beta 0 + \beta 1 \times IL + \beta 2 \times CBO$$

In this model, we model a class' complexity along two different dimension design concept. By combing CBO, we can model the design complexity of a class more precisely than using IL only. It is expected that the explanation ability of defect count of a class in the second model will be higher than the first model. Therefore, we have the second hypothesis.

H2: By using CBO and IL, the explanation ability of class defect count will be higher than using IL or CBO only.

We will test the second hypothesis by comparing the R-square values of these two models (1) and (2).

4 Experiment Analysis

There are several ways to empirically test the hypotheses. Firstly, small scale controlled experiments like homework assignment at school can be set up. Although in the small scale controlled experiment we may better control the factors which have impact on software quality, such as design complexity, student skill, development tools etc, the defect count of classes in simple project may have just little difference. Thus, it may be hard to validate the relationship between design complexity and class defect count. Secondly, the experiment setting in software industry should better reflect such relationship, but we do not have any available industry software data such as design document, the results of testing etc, to set up experiments. So, our approach is taking open source projects as experimental subjects. We focus on Apache [1] open source projects since the projects in Apache Software Foundation have more detailed documentation and every java class in a project has its own change log.

Information about each class of an open source project can be collected from its change log. However, adapting open source projects as experimental subjects has some threats to validity. We will discuss that at the end of this section.

4.1 Dependent and Independent Variables

The dependent variable is the number of defects of each Java class. Java interface is excluded from the data samples. The number of defects is defined as the number of revisions of each Java source class recorded as bug fixing. In the projects we study, when a revision of a Java class is bug fixing, the revision log will contain the string " PR: " or " bug " and some additional information about this revision. Here, " PR " means problem report, and it associates with the bugzilla bug database [2]. In this case, we count such revision as a defect. We need identify a particular time interval during which developers perform major bugs fixing activities. Defect count is collected during this time interval. The principal to identify this time interval will be described in the following section.

The independent variables are IL and CBO. It only needs class interface and attribute information to compute IL and Java reflection technology [12] can easily acquire them. So we use Java reflection APIs to implement the metric calculation tool. The IL value of a class is the summation of IL values of all method declared in the class, and inherited methods are not included. In our experiment, CBO value measurement is only based on the interface and attributes of a Java class. Although CBO definition usually includes not only the classes in the interface and instance variables but also the variables declared locally within the method, what we want to measure is the coupling relationship which can be identified at the earlier design phase the same as interaction level applies. And large portion of coupling relationship can be identified by class interface and instance variables. The small portion of inaccuracy only causes a little influence.

4.2 Experimental Subjects

We focus on the projects implemented with Java language and follow several principals to take projects as our experimental subjects. Firstly, the relationship we want to validate is design complexity and its influences on class defect count. We identify a more clear time interval during which major bug fix activities were performed for a particular release in the project history. The project document and change history help to make the judgement. For example, in Table 2, the first project is Jakarta-ORO. In this project, the interval we identify is from 2.0.0 release to 2.0.8 release. During this interval, there is other release, such as 2.0.2, 2.0.3 and so on. The changes documented between these releases contain many bug fixing activities. Since the releases following 2.0.0 contain many bug fixing activities and have the same major release number 2.0, we suggest these activities are performed for release 2.0. The second principal is the number of classes in a project should not change greatly during this time interval. The increasing number of classes in the project may imply that some other development activities performed to enhance or improve the functionality of the software, and the changes of design may generate other defects in the software. Therefore, we assume most of the bug fixing activities are caused by the design complexity of source release of the software we observed because the number of classes of the project does not change greatly in following releases (See Table 3).

The projects we used in this experiment are depicted in Table 2. The first column is the project's name. The second column is the release by which we measured the metrics values. The third column depicts the end release, we collect defect data between start release and end release. The forth column records the total time which we collected defects data.

4.3 Experimental Result

We build the linear regression models of the six projects and test whether a relationship exists between software metrics and class defect count. If there is a positive nonzero linear regression relationship exists, the coefficient of the independent variables will be larger than zero. Table 4 lists the linear regression models built by using only IL as independent variable. Table 5 lists the linear regression models built by using only CBO as the independent variable. And Table 6 lists the linear regression models built by using IL and CBO as the independent variables. The value followed by the coefficient value is the p-value of the corresponding parameter. The sixth columns in Table 6 are the growth of adjusted R-square value compared to the higher one of models using single metric only. The last column is the correlation between IL and CBO.

Table 2: Release Information of The Experimental Subjects

Project Name	Start Release	End Release	Total Time
Jakarta-ORO	2.0.0 (2000/7/23)	2.0.8 (2003/12/30)	40 months
Commons-HttpClient	3.0 alpha1 (2004/5/17)	3.0 RC2 (2005/4/9)	11 months
Jakarta-Velocity	1.0 beta1 (2001/3/20)	1.2 RC1 (2001/9/26)	6 months
Jakarta-POI	2.0 pre1 (2003/5/17)	2.0 rc1 (2003/11/2)	6 months
Jakarts-Struts	1.0 Beta1 (2001/2/23)	1.0.2 (2002/2/11)	12 months
Cocoon	2.1M1 (2003/4/29)	2.1.5.1 (2004/7/9)	14 months

Table 3: The Class Count of Start and End Release

Project Name	# classes of start release	# classes of end release
Jakarta-ORO	61	61
Commons-HttpClient	120	129
Jakarta-Velocity	150	176
Jakarta-POI	308	311
Jakarta-Struts	176	185
Cocoon	495	515

4.3.1 Influence of IL

From Table 4 and 6, the regression results indicate that increase in IL value associates with increase in defect count because the parameters of IL in six project regression models are positive. And the p-values of IL’s parameter are all smaller than 0.05. That means the influence of IL on class defect count is significant.

4.3.2 Influence of CBO

From Table 5, six multivariate regression models show that increase in CBO value associates with increase in defect count since the coefficients of CBO in these models are positive and significant with the p-values are smaller than 0.01. From Table 6, five out of six models show the similar result. The exception is Jakarta-ORO. In this project, the univariate analysis about CBO indicates the impact of CBO is positive and significant. But, in the multivariate regression model, the coefficient of CBO becomes negative. Besides, in the models of Jakarta-ORO and Jakarta-POI, the multiple regression analysis with two independent variables was performed to determine the explanatory ability of these variables. These models almost did not show any increase in adjusted R-square. All of these are symptoms of collinearity. We find that the correlation values between CBO and IL in these two models are relatively higher than others. IL and CBO account for most of the same variance in class defect count. That may be the reason for the adjusted R-square value did not increase. (Table 6: correlation for Jakarta-ORO is 0.741 and correlation for Jakarta-POI is 0.761).

4.3.3 Discussion of Result

Result of Hypothesis 1: From the observation of IL influence on defect count, it indicates a class has higher IL value will be associated with higher number of defects. In other words, the class complexity modeled by IL influence the class reliability. Thus IL can be an appropriate class complexity metric. The result supports our hypothesis one. **Result of Hypothesis 2:** From the observation of the CBO influence on defect count, four out of six multivariate regression models have growth of R-square value about or above fifteen percent. Although IL and CBO have some correlation, basically they model different dimension design properties. That is why the adjusted R-square value increases. Thus, this indicates CBO and IL are complementary class defect count indicators. The result supports our hypothesis two.

4.4 Threats to Validity

In the experiments, we want to investigate the relationship between design complexity and class defect count collected during a particular timing interval performing testing activities. However, some factors have effects on the experiment result. During the time interval we collect class defects, some other development activities are performed such as enhancement or update. Because we did not distinguish when the defects exist in the software and these activities may generate additional defects counted as the defects caused by design complexity before enhancement or update, these activities result in the inaccuracy of the relationship between design complexity and defect count. Design complexity of some projects are not first major release. For example, Commons-HttpClient is 3.0 alpha release 1. In these projects, some classes in the system are developed from the project initiation. They may be tested and corrected after the previous several major releases. The classes in the software system undergo different

Table 4: Linear Models Using IL as the Independent Variable

Project Name	Coefficient of Intercept	Coefficient of IL	Adjusted R-square
Jakarta-ORO	-2.158E-02 (.882)	6.1E-04 (.000)	.762
Commons-HttpClient	.734 (.000)	1.28E-04 (.000)	.365
Jakarta-Velocity	2.35 (.000)	3.26E-04 (.000)	.161
Jakarta-POI	.418 (.000)	9.7E-05 (.000)	.316
Jakarta-Struts	.635 (.000)	1.77E-04 (.000)	.248
Cocoon	.268 (.000)	6.14E-05 (.000)	.201

Table 5: Linear Models Using CBO as the Independent Variable

Project Name	Coefficient of Intercept	Coefficient of CBO	Adjusted R-square
Jakarta-ORO	-.255 (.395)	.707 (.000)	.309
Commons-HttpClient	.215 (.257)	.715 (.000)	.434
Jakarta-Velocity	1.55 (.000)	.922 (.000)	.182
Jakarta-POI	.230 (.000)	.348 (.000)	.415
Jakarta-Struts	.461 (.038)	.808 (.000)	.101
Cocoon	.121 (.016)	.260 (.000)	.166

Table 6: Linear Models Using IL and CBO as the Independent Variables

Project Name	Coefficient of	Coefficient of	Coefficient of	Adjusted	Growing	Correlation
Jakarta-ORO	.104 (.558)	6.68E-04	-.146 (.229)	.764	0	0.741
Commons	.273 (.123)	7.2E-05 (.000)	.505 (.000)	.510	15	0.569
Jakarta Velocity	1.66 (.000)	2.00E-04	.643 (.001)	.222	22	0.521
Jakarta-POI	.251 (.000)	3.00E-05	.276 (.000)	.425	2	0.761
Jakarta-Struts	.257 (.185)	1.60E-04	.559 (.001)	.293	18.1	0.222
Cocoon	.123 (.010)	4.54E-05	.164 (.000)	.253	25.8	0.467

development time, and the development time has effect on the class defect count. In our experiment, we regard the development time of each class as the same. But During the time interval we identify, more defects may be founded in newly defined or modified classes than the classes tested before and to be fit in new context.

5 Conclusion

Firstly, we empirically investigate the relationship between IL metric and class defect count. The results seem to show IL can be an indicator of the class defects during design phase. Secondly, by combing CBO and IL to model the design complexity of a class, there is higher explanation ability of the empirical models to explain class defect count. This implies the quality of a class can be modeled in different dimensions software design. Using design complexity metrics would be able to help developers to make design inspection more efficient and provide useful information to developers.

Acknowledgments

This research is supported by the Taiwan National Science Council (NSC) under contract NSC-94-2213-E-006-075.

References

[1] Apache Software Foundation. http://www.apache.org

[2] Apache Software Foundation Bug System. http://issues.apache.org/bugzilla

[3] D. H. Abbott, T. D. Korson, and J. D. McGregor, "A design complexity metric for object-oriented development," Tech. Rep. 94-105, Department of Computer Science, Clemson University, 1994

[4] R. K. Bandi, V. K. Vaishnavi, and D. E. Turk, "Predicting maintenance performance using object oriented design complexity metrics," *IEEE Transaction on Software Engineering*, vol. 29, no. 1, Jan. 2003.

[5] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Transactions on Software Engineering*, vol. 22, no. 10, pp. 751-781, Oct. 1998.

[6] L. C. Briand, J. Wust, S. V. Ikonomovski, and H. Lounis, "Investigating quality factors in object oriented designs: An industrial case study," *Proceedings of the 21st international conference on Software engineering*, pp. 345-354, May 1999.

[7] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 8, pp. 476-493, June 1994.

[8] G. K. Gill and C. F. Kemerer, "Cyclomatic complexity density and software maintenance productivity," *IEEE Transactions on Software Engineering*, vol. 17, no. 12, pp. 1284-1288, Dec. 1991.

[9] Java Reflection Technology. http://java.sun.com/j2se/1.3/docs/guide/reflection/

[10] W. Li and S. Henry, "Object-oriented metrics that predict maintainability," *Journal of Systems and Software*, vol. 23, no. 2, pp. 111-122, Nov. 1993.

[11] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 308-320, Dec. 1978.

[12] V. Y. Shen, T.-J. Yu, S. M. Thebaut, and L. R. Paulsen, "Identifying error-prone software -an empirical study," *IEEE Transactions on Software Engineering*, vol. 11, no. 4, pp. 317-324, April 1985.

[13] I. Sommerville, *Software Enginnering*. Addison-Wesley, 6 ed., 2000.

[14] R. Subramanyam and M. Krishnan, "Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects," *IEEE Transactions on Software Engineering*, vol. 29, no. 4, pp. 297-310, April 2000.