Class Cohesion as predictor of changeability: An Empirical Study

Hind Kabaili — Rudolf Keller — François Lustman

Département IRO
Université de Montréal
C.P. 6128, succursale Centre-ville
Montréal, Québec H3C 3J7, Canada

kabaili@iro.umontreal.ca

keller@iro.umontreal.ca

lustman@iro.umontreal.ca

ABSTRACT: *The assessment of the changeability of software systems is of major concern for buyers of the large systems found in fast-moving domains such as telecommunications. One way of approaching this problem is to investigate the dependency between the changeability of the software and its design, with the goal of finding design properties that can be used as changeability indicators. In this research, we set out to investigate first, the relationship between cohesion and coupling, then the relationship between cohesion and impacts of change. The data collected from three test systems of industrial size indicate no such correlation. Suspecting that cohesion metrics adopted for the experiment do not adequately capture the cohesion property, we analyzed the classes with lowest cohesion.*

RÉSUMÉ: *L'évaluation de la changeabilité des logiciels intéresse tout particulièrement les acheteurs de grands systèmes dont le domaine d'application évolue en permanence, comme c'est le cas en télécommunication. Une manière d'aborder le problème serait d'étudier la dépendance entre la changeabilité des logiciels et leur conception, afin de trouver des propriétés architecturales pouvant être utilisées comme indicateurs de changeabilité. Dans le travail rapporté ci-dessous, nous avons d'abord étudié la relation entre la cohésion et le couplage, puis la relation entre la cohésion et des impacts de changements. Les données collectées à partir de trois systèmes de test, de taille industrielle, ne montrent aucune corrélation. Nous avons analysé les classes faiblement cohésives car il semblait que les métriques de cohésion ne capturaient pas adéquatement la propriété de cohésion.*

KEY WORDS: *changeability, design metrics, object-oriented, cohesion, coupling, empirical validation.*

MOTS-CLÉS: *changeabilité, métriques de conception, orienté objet, cohésion, couplage, validation empirique.*

1. Introduction

The use of object-oriented (OO) technology for developing software has become quite widespread. Researchers assert that OO practice assures good quality software. By quality software, they mean maintainable, reusable, and easily extensible. Industrial buyers want to be sure of the quality of the product they acquire. For this, they need OO measures, to evaluate the software they want to buy.

For various reasons, Bell Canada, the industrial partner in this project[1], is interested in buying large-scale software rather than developing it. It needs to be sure of the quality of the systems it acquires. As part of the project, design properties are investigated as changeability indicators.

To assess with some objectivity the quality of a design, we need to quantify design properties. The most known and most used design properties in OO design are coupling and cohesion. In the realm of OO systems, experiments have been conducted, showing that coupling between classes is an indicator of changeability. Chaumun et al. observed a high correlation between changeability and some coupling metrics, across different industrial systems and across different types of change [CHA 00]. However, measuring coupling is difficult and time consuming since it is an inter-class property. In fact, to measure it, the knowledge of the whole system and of all links between classes must be mastered. Cohesion is an intra-class property; to measure it we only need to consider the studied class. Also, a widely held belief in the design community, states that high cohesion is related to low coupling. Because of this supposed relationship, we decided to investigate cohesion as a changeability indicator.

Module cohesion was introduced by Yourdon and Constantine as “how tightly bound or related the internal elements of a module are to one another” [YOU 79]. A module has a strong cohesion if it represents exactly one task of the problem domain, and all its elements contribute to this single task. They describe cohesion as an attribute of design, rather than code, and an attribute that can be used to predict reusability, maintainability, and changeability. However, these assumptions have never been supported by experimentation.

In this work, we try to find out if cohesion can be used to predict the changeability of an OO system. By changeability of a system, we mean its capacity to absorb changes. Due to the belief that cohesion and coupling are related, we decided to correlate these two design properties. By showing this correlation, we will be able to assert that cohesion is a changeability indicator, too. Given the negative result of this shortcut experimentation, we conduct a second experimentation. A change impact model was used to calculate the changeability of the test systems, and thus, to test the relationship between cohesion and impact of change.

[1] This research was supported by the SPOOL project organized by CSER (Consortium for Software Engineering Research) which is funded by Bell Canada, NSERC (National Sciences and Research Council of Canada), and NRC (National Research Council of Canada).

The paper is organized as follows. Section 2 presents an overview of cohesion as a design property. Section 3 describes the potential relationship between cohesion and coupling, and the empirical validation. Because of a lack of correlation in the shortcut experimentation, we conducted a direct test between cohesion and changeability and reported it in section 4. The negative result of the two experimentations led us to investigate the reasons behind this lack of relationship. This investigation is described in Section 5. Section 6, finally, summarizes the work and provides an outlook into future work.

2. Cohesion and cohesion metrics

There is a consensus in the literature on the concept of class cohesion. A class is cohesive if it cannot be partitioned into two or more sets defined as follows. Each set contains instance variables and methods. Methods of one set do not access directly or indirectly variables of another set. Many authors have implicitly defined class cohesion by defining cohesion metrics. In the OO paradigm, most of the cohesion metrics are inspired from the LCOM metric defined by Chidamber and Kemerer (C&K) [CHI 94]. According to these authors "if an object class has different methods performing different operations on the same set of instance variables, the class is cohesive". As a metric for assessing cohesion, they define LCOM (Lack of Cohesion in Methods) as the number of pairs of methods in a class, having no common attributes, minus the number of pairs of methods having at least one common attribute. The metric is set to zero when the value is negative.

Li and Henry [LI 93] redefine LCOM as the number of disjoint sets of methods accessing similar instance variables.

Hitz and Montazeri [HIT 95] restate Li's definition of LCOM based on graph theory. LCOM is defined as the number of connected components of a graph. A graph consists of vertices and edges. Vertices represent methods. There is an edge between 2 vertices if the corresponding methods access the same instance variable. Hitz and Montazeri propose to split a class into smaller, more cohesive classes, if LCOM > 1.

Bieman and Kang [BIE 95] propose TCC (Tight Class Cohesion) and LCC (Loose Class Cohesion) as cohesion metrics, based on Chidamber and Kemerer's approach. They too consider pairs of methods using common instance variables (constructors and destructors are not taken into account). However, the way in which they define attribute usage is different. An instance variable can be used directly or indirectly by methods. An instance variable is directly used by a method *M*, if the instance variable appears in the body of the method *M*. The instance variable is indirectly used, if it is directly used by another method *M'* which is called directly or indirectly by M. Two methods are directly connected if they use directly or indirectly a common attribute. Two methods *M* and *M'* are indirectly connected if *M* is directly connected to *M''* and *M''* is directly connected to *M'*. TCC is defined as the percentage of pairs of methods that are directly connected. LCC counts the pairs

of methods that are directly or indirectly connected. We recall that constructors and destructors are not taken into account for computing LCC and TCC. The range of TCC and LCC is always in the [0,1] interval. They propose three ways to calculate TCC and LCC: (1) include inherited methods and inherited instance variables in the analysis, (2) exclude inherited methods and inherited instance variables from the analysis, or (3) exclude inherited methods but include inherited instance variables. In respect to the three ways of calculating their metrics, Bieman and Kang do not express any preference. We opted for evaluating them according to the first way, considering inheritance as an intrinsic facet of OO systems. LCC is an extension of TCC in that additional features are taken into account. LCC being more comprehensive than TCC, we adopted LCC, together with LCOM (original C&K version), as the prime cohesion metrics of our experimentation.

3. Relationship between cohesion and coupling

As a principle of good OO design, the components of a class should contribute to one specific task. A non-cohesive class means that its components tend to support different tasks. According to common wisdom, this kind of class has more interactions with the rest of the system than classes encapsulating one single functionality. Thus, the coupling of this class with the rest of the system will be higher than the average coupling of the classes of the system. This relationship between cohesion and coupling means that a non-cohesive class should have a high coupling value. But in spite of the widely-held belief in this relationship, it has never been thoroughly investigated. However, the coupling property has extensively been studied. There is coupling between two classes if one class access or uses some element of the other class.

Many metrics that capture interactions between classes have been defined. Chidamber and Kemerer proposed two coupling metrics [CHI 94] that have been validated as fault prone indicators [BAS 96]:

CBO (*Coupling between Object Classes*): A class is coupled to another one if it uses its member functions and/or instance variables, and vice versa. CBO provides the number of classes to which a given class is coupled.

RFC (*Response for a Class*): This is the number of methods that can potentially be executed in response to a message received by an object of that class.

Briand et al. describe coupling as the degree of interdependence among the components of a software system. They defined 18 coupling metrics. This suite takes into account the different OO design mechanisms provided by the C++ language [BRI 97].

While the relationship between cohesion and quality has not been quantitatively assessed, several coupling metrics have been shown to be good quality indicators with respect to some specific quality aspects. We decided to investigate the potential of cohesion metrics as changeability indicators by looking for relationships between cohesion and coupling.

3.1. Selection of metrics

To test our hypothesis "low cohesion is correlated with high coupling", we adopted some well-known cohesion and coupling metrics found in the literature. As cohesion metrics, we chose LCC and LCOM (see Section 2.1). For measuring coupling, we adopted CBO and RFC, since these two metrics have been proven to be good indicators of quality [BAS 96] and changeability [CHA 99, CHA 00]. To assess our hypothesis empirically, the following correlation hypotheses must be tested statistically:

- For the test system, there is a relationship between the LCC and CBO metrics.
- For the test system, there is a relationship between the LCC and RFC metrics.
- For the test system, there is a relationship between the LCOM and CBO metrics.
- For the test system, there is a relationship between the LCOM and RFC metrics.

Thus, in our experiment, we attempted to correlate the LCC and LCOM metrics with the C&K coupling metrics (CBO, RFC) and extend the scope of the LCC and LCOM metrics to the changeability property. During experimentation, we decided to include in our study the NOC (number of children) metric which is usually considered as a coupling metric. Furthermore, we considered four metrics that we derived from the NOC and CBO metrics. Recall that CBO is "approximately equal to the number of coupling with other classes (where calling a method or instance variable from another class constitutes coupling)" [CHI 98]. Below, we present the four metrics, together with the rationale for their consideration

- *NOC** (*Number Of Children in subtree,* also called number of successors in [ABR 93]): when some component of a class is changed, it may affect not only its children but also the whole subtree of which the changed class is the root.
- *CBO_NA* (*CBO No Ancestors: same as CBO, but the coupling between the target class and its ancestors is not taken into consideration*): the coupling between the class and its ancestors, taken into consideration by CBO, is irrelevant for change impact, since the ancestors of the target class will never be impacted. To eliminate such "noise", ancestors are excluded in CBO_NA.
- *CBO_IUB* (*CBO Is Used By: the part of CBO that consists of the classes using the target class*): the definition of CBO merges two coupling directions: classes using the target class and classes used by the class. For changeability purposes, the former seems more relevant than the latter one, hence the split.
- *CBO_U* (*CBO Using: the part of CBO that consists of the classes used by the target class*): introduced as a consequence of CBO_IUB, to cover the part of CBO not considered by CBO_IUB.

In summary, seven metrics were considered: the two C&K coupling metrics (CBO, RFC), one other C&K design metric (NOC) and four changeability-oriented refinements of the C&K metrics suite (NOC*, CBO_NA, CBO_IUB, CBO_U).

To achieve significant and general results, the data used to test the correlation between cohesion and coupling were collected from three different industrial OO systems, as described below.

3.2. Environment

In this section, we first present the three test systems of the experiment. Then, the environment in which the experiment was conducted is described. Finally, we discuss the experimental procedure that was adopted.

The considered systems were chosen to meet the objectives of the SPOOL project. They are industrial systems, they vary in size and the domain of at least one of them belong to our research partner area.

The first test system is *XForms*, which can be freely downloaded from the web [XFO 97]. It is a graphical user interface toolkit for X window systems. It is the smallest of the test systems (see Table 1). *ET++*, the second test system, is a well-known application framework [WEI 89]. The version used in the experiment is the one included in the SNiFF+ development environment [TAK 99]. The third and largest test system was provided by *Bell Canada*, and is called, for confidentiality reasons, *System-B*. It is used for decision making in telecommunications. Table 1 provides some size metrics for these systems.

	Xforms	*ET++*	*System-B*
Lines of code	7 117	70 796	291 619
Lines of pure comments	764	3 494	71 209
Blank lines	1 009	12 892	90 426
# of effective classes	83	584	1 226
# of methods	450	6 255	8 594
# of variables	1 928	4 460	13 624
Size in repository	2.9 MB	19.3 MB	41.0 MB

Table1. *Size metrics of test systems*

To calculate the metrics involved in the experimentation, we used the SPOOL environment (see Figure 1). This environment is being developed for the entire SPOOL project and comprises various analysis and visualization capabilities to cope with large-scale software systems [KEL 99].

The environment provides a repository-based solution. A parsing tool, e.g., a compiler front-end, parses the test system source code. GEN++, the C++ implementation of *GENOA* [DEV 92], was used in this extraction process. The parsed information contains data about all classes and links in the system. This information is captured and fed into a design repository. The schema of the design repository is based on our extended UML (Unified Model Language) metamodel [RUM 99]. The OO database management system *POET 5.1* [POE 99] serves as the repository backend, with the schema being represented as a Java 1.1 class hierarchy. Metrics requests are batch-processed using a flexible report generator mechanism. They typically contain information on the metrics as well as on the target class, methods, and variables. This triggers a set of queries corresponding to the specified metrics. The code in these queries uses the metrics request information as parameters to interrogate the repository. Raw results are fetched and processed into ASCII files that obey a specific format and can readily be transferred into spreadsheet programs such as Excel for further statistical processing.

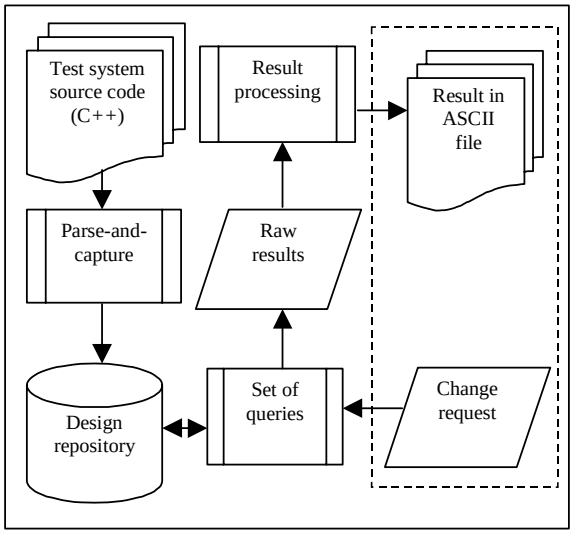

Figure 1. *Environment for metrics calculation*

We collected cohesion metrics values from the three test systems. Furthermore, we gathered the values for all seven metrics explained in Section 3.2. For each metric involved in the experimentation, we calculated some descriptive statistics (minimum, maximum, mean, median, and standard deviation; see Appendix A and B). To test the relationship between the cohesion and the coupling metrics we used the Pearson correlation coefficient (see Appendix C). This statistical technique is

widely used for measuring the degree of relationship between two variables [SPI 00].

3.3. Results

Descriptive statistics of the three test systems are summarized in Appendices A and B. NOC and NOC* have the same median value for the three systems, 0 for NOC and NOC*. A median of 0 for number of children (NOC) and for NOC* means that for the three systems, half the classes are leaves. Based on this and on the mean value of NOC, it can be stated that classes that do have children have on the average less than two children. These results were found in software systems of different size and application domain, and we conclude that in general inheritance is not strongly used in OO development. Thus, the class trees of such systems will generally be flat.

According to C&k [CHI 94] and Bieman and Kang [BIE 95], a class is strongly cohesive when LCC ≈ 1 or LCOM = 0. Appendix A shows the mean value for both LCC and LCOM. Based on the mean value of LCC and LCOM, μ(LCC) = 0.62 and μ(LCOM) = 5.81, we can conclude that Xforms classes are not so strongly cohesive. For ET++, μ(LCC) = 0.42 and μ(LCOM) = 89.70. Finally for System-B, μ(LCC) = 0.56 and μ(LCOM) = 145.73. Based on these values, and referring to the definition of both LCOM and LCC, we concluded that the three test systems classes are not strongly cohesive.

Note that a similar reasoning can be done based on the median value of both LCC and LCOM.

According to the median value of LCOM, half the classes may be split for Xforms, and more than half for ET++ and System-B. On the other hand the median values of LCC, for Xforms (0.69) and System-B (0.61), suggest that half the classes have a LCC value bigger that 0.6. This means that half the classes are acceptably cohesive. At this stage we can conclude that there is discrepancy between LCC and LCOM.

The Pearson correlation coefficients are presented in Appendix C. According to these coefficient values, no correlation can be claimed between the LCC and LCOM metrics and CBO and RFC (we also checked for outliers). Moreover, no correlation was found between the tested cohesion metrics and the five other metrics of the study. No general conclusion was drawn at this stage. However, these negative results were found for two cohesion metrics and seven coupling metrics (CBO, RFC, NOC, NOC*, CBO_NA, CBO_IUB, CBO_U), across three industrial systems of different size and origin. Therefore, we put forward the following hypothesis: in general, there is no relationship between these cohesion metrics and coupling metrics.

It is early to conclude from the negative results of this experimentation, that cohesion is not a changeability indicator. We conducted a second experimentation to correlate cohesion and changeability through a change impact model.

4. Relationship between cohesion and impacts of change

4.1 change impact model

One way assessing the changeability of an OO system is by performing a change impact analysis. In this work, the changeability of OO software is assessed by an impact model defined in [CHA 98] [CKK 99]. Below we detail the changes considered and the links involved, and we introduce the notions of impact and impact expression.

Changes

A change applies to a class, a variable or a method. Examples are deleting a variable, changing the signature of a method, or removing a class from the list of parents of another class. Thirteen changes have been identified:

Variable: addition, deletion, type change, and scope of change.

Method: addition, deletion, return type change, implementation change, signature change, and scope change.

Class: addition, deletion, and inheritance relationship change.

The changes considered in this paper are atomic changes. More complex changes, for instance refactoring operations such as moving a variable or a method along the class hierarchy, or inserting a new class to factor out some common characteristics of a group of classes, will be addressed in future work. In this way, changes can be dealt with at a higher level of abstraction.

Links

The following links connect classes one to another. They reflect usual connections in OO systems, and are not specific to any particular OO programming language.

S (*association*): one class references variables of another class.

G (*aggregation*): the definition of one class involves objects of the other class.

H (*inheritance*): one class inherits the features defined in another (parent) class.

I (*invocation*): methods in one class invoke methods defined in another class.

Any number and type of links between two classes may be found. Note that instantiation is not a link in its own right, but is taken into account with the invocation link. A change in a class may also have an impact in the same class. The pseudo-link **L** (*local*) is introduced to express this. We also consider a special notation commonly used in the Boolean algebra; the prime after a link, means the set of classes not associated with that special link. For example **G'** means the set of classes that are not associated with the aggregation link to the specified class.

Impact

We call *impact of a change* the set of classes that require correction as a result of that change. It depends on two factors. One is the type of change. For example, a

change to a variable type has an impact on all classes referencing this variable whereas the addition of a variable has no impact on those classes. Given a type of change, the other factor is the nature of the links involved. If, for instance, the scope of a method is changed from public to protected, the classes that invoke the method will be impacted, with the exception of the derived classes. Note that we limit ourselves to syntactic impact; considering semantic impact, for instance runtime errors, is beyond the scope of this paper. The impact of change ch_j to class cl_i is defined by a set expression E in which the variables are the sets defined by the various links:

Impact(cl_i, ch_j) = E (**S**, **G**, **H**, **I**, **L**)

For example,

Impact(cl_i, ch_j) = **SH'+G**

means that the impacted classes are those associated (**S**) with, but not inheriting (**H'**) from cl_i or those aggregated (**G**) with cl_i.

4.2. Application to C++

The industrial partner of our project was interested in the evaluation of programs in C++ for which only the code was available. The model was therefore mapped into that language.

In the C++ model, a change is a syntactic change to the code, and impact is considered if as a result of the change, the code at some other place, does not recompile successfully. The links identified in the conceptual model exist at the code level, and an additional one, **F** for *friendship,* is added to reflect the existence of this feature in C++. Changes were enumerated and for each, the impact set-expression was derived by examining all possible combinations of links between a changed class and another class. As an example, the change in a variable scope from public to private (code change from `public int v;` to `private int v;`) results in the impact **SF',** meaning that the impacted classes are those linked to the changed class by association but not by friendship. The 13 changes defined at the conceptual level expanded to 66 changes in C++, and their impact expression was compiled, 12 for variables, 35 for methods, and 19 for classes (see [CHA 98] for the list of changes and impact calculations).

4.3. Empirical validation

To test the hypothesis that cohesion is correlated to changeability, we adopted the same cohesion metrics, LCC and LCOM used in the first experimentation (see Section 2.1).

For lack of resources, we were unable to investigate the whole list of 66 changes of our impact model for C++ (see Section 2.3). Rather, we limited ourselves to six

changes, which we selected according to four criteria. First, there should be at least one change for each component (variable, method, and class). Second, a selected change should indeed have an impact in at least one other class (according to our model, there are 29 changes with no such impact). Third, the impact expression should be different for any pair of changes; since otherwise, we would have obtained duplicate results. And fourth, as an informal criterion, we required the selected changes to be of practical relevance, that is, they should be suitable to be exercised in practice. Table 2 lists the six changes considered and their corresponding impact expressions.

	Change	**Impact Expression**
1.	Variable type change	**S + L**
2.	Variable scope change from public to protected	**SH'F'**
3.	Method signature change	**I + L**
4.	Method scope change from public to protected	**H'IF'**
5.	Class derivation change from public to protected	**H'F' (S + I)**
6.	Addition of abstract class in class inheritance structure	**S + G + H + I + L**

Table 2. *Investigated changes with impact expressions*

In the experiment, we first extracted the LCC and LCOM metrics from the test systems. Next, for each of the six changes considered, and each of the test systems, we determined its test set, that is, the set of classes for which the change is applicable. For example, when considering the method scope change from public to protected (Change #4), only classes with at least one public method were included in the test set. Then, for each class in each test set, the change impact for the given change was calculated, i.e., the number of classes that would be impacted. If the change was one that affected a variable or method component (Changes #1 through #4), the change impacts for each individual variable or method of the given class were added together, and the total was divided by the number of variables or methods in the class.

Once the metrics and impact data were collected, we investigated the correlation between each change impact and each design metric for all the classes involved in the test sets. Then, in each case the correlation coefficient was calculated.

4.4. Results

Each of the six changes was applied to each test system. The impact values are presented in Appendix D.

The values vary from one system to another, from one change to another, and no general conclusion can be drawn on the impact of a given change. Comparison between changes, however, yields some results. Based on both mean values and median values, a classification of changes by impact comes out. Among the six changes investigated, the most expensive one, across systems, is the addition of an abstract class in the inheritance structure of a class (Change #6). On the other hand, the least expensive one is to change the scope of a method from public to protected (Change #4). This might have been expected, considering their impact expressions.

The Pearson correlation coefficients are presented in Appendix E. Two exceptions aside, most correlation coefficients for the two cohesion metrics are weak. The two exceptions are, for Xforms, the correlation coefficients between LCC and the change #1 and between LCC and change #5 are around 0.5. However, there are not significant enough to confirm the correlation hypothesis.

5. Investigation

The goal of our study was to find a correlation between cohesion and changeability, but the result was negative. Consequently, we set out to reason about this absence of correlation. We came up with the following two explanations: (1) the cohesion or coupling metrics chosen for the experimentation or the impact of changes are not the right ones, (2) there is no relationship between cohesion and changeability. Explanation (2), being counter to a widely held belief in the design community, was discarded. We then focused our investigation on hypothesis (1). We derived from (1) the following sub-hypotheses:

(1A) The LCC and LCOM metrics do not correctly measure cohesion.

(1B) The CBO and RFC metrics do not correctly measure coupling.

(1C) The change impact model is not correct.

The coupling metrics were validated in [HAR 98], and widely used with no major problem reported [BAS 96, CHA 00], thus sub-hypothesis (1B) was rejected. Sub-hypothesis (1C) was also rejected on the grounds that the change impact model is quite well validated [CHA 99].

On the other hand, we questioned the quality of the investigated cohesion metrics (sub-hypothesis (1A)). Intuitively, when they show a high class cohesion (LCC =1 or LCOM = 0), the classes are probably quite cohesive. However, we were doubtful about the expressiveness of LCOM and LCC in the presence of weak class cohesion. Thus, we set out to study manually various weakly cohesive classes occurring in the three test systems.

5.1. Study of weakly cohesive classes

According to the cohesion concept, a weakly cohesive class is designed in an ad hoc manner, and unrelated components are included in the class. The class represents several disparate concepts and may be split into several classes, each one modelling only one single concept. Based on anecdotal evidence, we suspected that, although LCC and LCOM indicate weak cohesion, it might not necessarily be true that the classes at hand must be broken into smaller components. To validate or reject this idea, we decided to manually inspect weakly cohesive classes.

We chose from each of the three test systems, classes that exhibit weak cohesion (LCC < 0.5 and/or LCOM > 0), to verify if they are real candidates for splitting. After studying these classes, we found that many of them should not be split. We came up with four major reasons for not splitting them.

First, some classes had no variables or only abstract methods, yielding low LCC values (and positive LCOM values).

Second, we noticed that for some classes, the LCC value is reduced by counting inherited variables or inherited values. For these cases, we calculated LCC without taking into account inherited components, and not surprisingly, we obtained LCC values indicating stronger class cohesion.

Third, some classes have multiple methods that share no variables but perform related functionalities. Putting each method in a different class would be counter to good OO design and the very idea of cohesiveness.

Fourth, we identified several classes that have numerous attributes for describing internal states, together with an equally large number of methods for individually manipulating these attributes.

Based on this analysis, we notice that low values of LCC and high values of LCOM do not assure a weakly cohesive class. We conclude that as measured, LCC and LCOM do in general not reflect the cohesion property of a class.

5.2. Additional cohesion properties

The results obtained in our study call for a refinement of the definition of cohesion metrics, in order to better measure the cohesion property as stated by OO design principles. It is our belief that a true cohesion metric will have to go beyond the simple sharing of class variables and capture additional information.

Briand et al. provide a categorization of cohesion metrics [BRI 98]. LCOM is counted as a cohesion metrics based on common attribute usage in a class. LCC belongs to the cohesion metrics category that is based on both common attribute usage and method invocations within a class.

Chae and Kwon, in their recent paper, reflect on the weakness of current research on class cohesion measures [CHE 98]. They observe that existing approaches do not consider the special methods that interact with only part of the instance variables

and thus reduce class cohesion. As examples, they mention accessor methods, delegation methods, constructors, and destructors. They propose that special methods be treated such that they do not compromise the value of the cohesion metrics. Furthermore, Chae and Kwon suggest that cohesion metrics take into account additional characteristics of classes, for instance, the patterns of interaction among the members of a class. Their reasoning about special methods confirms the fourth reason we brought up in the previous section.

We believe that this work clearly constitutes an improvement in calculating class cohesion. However, it is our contention that we must take into account not only the patterns of interaction among class members, but also the semantics of these interactions. Based on our investigation results, we furthermore assert that cohesion measures must take into account the functionality of class methods as well as the unity of the data that describe the entity modeled by the class.

6. Conclusion

In this paper, our major goal was to validate cohesion metrics as changeability indicators. To this end, we tried to correlate cohesion metrics with coupling metrics that had been proven as quality indicators. We chose LCC and LCOM as cohesion metrics, and CBO and RFC were chosen as the primary coupling metrics. We collected data about these metrics on three different industrial systems. Our experimentation showed no correlation between cohesion and coupling metrics chosen. A second experimentation was conducted to correlate cohesion metrics with impact of change. First, a model of software changes and change impact was adapted for C++ language. For practical reasons, we only investigated six changes, chosen to be representative of C++ systems changes. Furthermore, we limited our definition of change impact to recompilation errors. The negative results of the two experimentations led us to think about reasons of the lack of correlation.

We suspected that the cohesion metrics used in the experimentation do not reflect the real cohesion of a class. We decided to investigate manually classes with low cohesion metric values. We found that although some classes have low LCC and/or high LCOM, these classes are actually cohesive.

A cohesion measure based on the variable sharing aspect is a special way of capturing class cohesion. This restricted definition led to cohesion measures with misleading values in several situations. Such situations occur, for instance, when classes have abstract methods or when a class inherits a large number of methods or instance variables from its superclass. When taking into account these abstract methods or inherited components, the cohesion value of a class is reduced, resulting in misleading class cohesion values. In our belief, class cohesion metrics should not exclusively be based on common attribute usage and method invocation, but also on patterns of interaction between class members, on the functionality of class methods, and on the conceptual unity of its instance variables.

Based on these results we came up with two conclusions. The original idea that cohesion is easy to measure must be rejected. And, as long as a new cohesion metric is not defined, taking into account important facets of the cohesion property, actually defined cohesion metrics cannot be trusted as changeability indicators. As future work, we are trying to feed our database with new test systems. In the same direction, we project to extend the change impact model.

7. References

[ABR 93] ABREU F. B., "Metrics for Object-Oriented Environment", *In Proceedings of the Third International Conference on Software Quality*. Lake Tahoe, Nevada, October 4-6, 1993, p. 67-75.

[BAS 96] BASILI V., BRIAND L. C., MELO W. L., "A validation of object-oriented design metrics as quality indicators", In *IEEE Transactions on Software Engineering*, vol. 22 no 10, p. 751-761, October 1996.

[BRI 97] BRIAND L., DEVANBU P., MELO W. L., " An investigation into coupling measures for C++", In *Proceedings of the International Conference on Software Engineering (ICSE'97)*, p. 412-421, Boston, MA, May 1997.

[BIE 95] BIEMAN J. M., KANG B. K., "Cohesion and reuse in an object-oriented system", In *Proceedings of the Symposium on Software Reusability (SSR'95)*, p. 259-262, Seattle, WA, April 1995.

[BRI 98] BRIAND L. C., DALY J., WUST J., "A unified framework for cohesion measurement in object-oriented systems", In *Empirical Software Engineering - An International Journal*, vol. 3 no 1, p. 67-117, 1998.

[CHA 98] M. A. Chaumun. Change impact analysis in object-oriented Systems: Conceptual Model and Application to C++. *Master's thesis*, Université de Montréal, Canada, November 1998.

[CHA 99] Chaumun M. A., Kabaili H., Keller R. K., Lustman F. "A change impact model for changeability assessment in object-oriented systems", In *Proceedings of the Third Euromicro Working Conference on Software Maintenance and Reengineering*, p. 130-138, Amsterdam, The Netherlands, March 1999.

[CHA 00] CHAUMUN M. A., KABAILI H., KELLER R. K., LUSTMAN L.,ST-DENIS G., "Design properties and object-oriented software changeability", *In Proceedings of the Fourth Euromicro Working Conference on Software Maintenance and Reengineering*, p. 45-54, Zurich, Switzerland, February 2000. IEEE.

[CHE 98] CHAE H. S., KWON Y. R., "A cohesion measure for classes in object-oriented systems", In *Proceedings of the Fifth international Software Metrics Symposium*, p. 158-166, Bethesda, MD, November 1998.

[CHI 94] CHIDAMBER S. R., KEMERER C. F., "A Metrics Suite for Object Oriented Design", In *IEEE Transactions on Software Engineering*, vol. 20, no. 6, p. 476-493, June 1994.

[CHI 98] CHIDAMBER S. R., DARCY D. P., KEMERER C. K., "Managerial use of metrics for object-oriented software: An exploratory analysis", In *IEEE Transactions on Software Engineering*, vol. 24 no 8, p. 629-639, August 1998.

[DEV 92] DEVANBU P. T., "GENOA - a customizable, language- and front-end independent code analyzer", In *Proceedings of the 14th International Conference on Software Engineering (ICSE'92)*, p. 307-317, Melbourne, Australia, 1992.

[HAR 98] Harrison R., Counsell S.J., Nithi R. , "An Investigation into the Applicability and Validity of Object-Oriented Design Metrics", *In Empirical Software Engineering: An International Journal*, , p. 255-274, vol. 3 no 3, 1998.

[HIT 95] HITZ M., MONTAZERI B., "Measuring coupling and cohesion in object-oriented systems", *Proc. Int. Symposium on Applied Corporate Computing*, p. 25-27, October, 1995.

[KEL 99] KELLER R. K., SCHAUER R., ROBITAILLE S., Page P., "Pattern-based reverse engineering of design components", *In Proceedings of the Twenty-First International Conference on Software Engineering*, p. 226-235, Los Angeles, CA, May 1999.

[LI 93] LI W., HENRY S., "Object-oriented metrics that predict maintainability", In *Journal of Systems and Software*, vol. 23, p. 111-122, February, 1993.

[POE 99] POET SOFTWARE CORPORATION, San Mateo, CA. POET Java ODMG Binding. Online documentation, 1999. Available online at <http://www.poet.com/>.

[RUM 99] RUMBAUGH J., JACOBSON I., BOOCH G., *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1999.

[SPI 00] SPIKARD J., *Understanding Correlation By Looking At Crime Rates*, Available online at: <http://www.mcguire-spickard.com/Fielding/ThinkStat/CorrPear.htm>, 2000.

[TAK 99] TAKEFIVE GMBH, Salzburg, Austria. SNiFF+ Documentation Set, 1999. Available online at: <http://www.takefive.com>.

[WEI 89] WEINAND A., GAMMA E., MARTY R., "Design and implementation of ET++, a seamless object-oriented application framework", In *Structured Programming*, vol. 10, no. 2, p. 63-87, April-June, 1989.

[XFO 97] XFORMS LIBRARY. Graphical user interface for X. Documentation Set, 1997. Available online at <http://bragg.phys.uwm.edu/xforms>.

[YOU 79] YOURDON E., CONSTANTINE L. L., *Structured Design*, Prentice Hall, Englewood Cliffs, N.J., 1979.

Appendix A: Cohesion metrics results for the test systems

System		**LCC**	**LCOM**
XForms 83 classes	**Minimum**	0	0
	Maximum	1	208
	Mean	0.62	5.81
	Median	0.69	1
	Std. Dev.	0.27	25.40
ET++ 584 classes	**Minimum**	0	0
	Maximum	1	4714
	Mean	0.42	89.07
	Median	0.33	6
	Std. Dev.	0.31	352.81
System-B 1226 classes	**Minimum**	0	0
	Maximum	1	11706
	Mean	0.56	145.73
	Median	0.61	10
	Std. Dev.	0.31	695.72

Appendix B: Coupling metrics results for the test systems

System		**NOC**	**NOC***	**CBO**	**CBO_ NA**	**CBO_ IUB**	**CBO_ U**	***RFC***
XForms 83 classes	**Minimum**	0	0	0	0	0	0	0
	Maximum	14	60	20	20	19	9	45
	Mean	0.82	2.57	4.13	3.16	0.98	3.16	6.52
	Median	**0**	**0**	4	3	0	4	2
	Std. Dev.	2.34	9.57	3.16	3.16	3.05	1.95	9.85
ET++ 584 classes	**Minimum**	0	0	0	0	0	0	0
	Maximum	56	361	301	301	293	76	746
	Mean	0.78	2.09	24.48	22.5	5.01	19.80	90.65
	Median	**0**	**0**	24	21.5	0	21	36.5
	Std. Dev.	3.45	17.05	25.40	24.63	21.28	15.89	128.98
System-B 1226 classes	**Minimum**	0	0	0	0	0	0	0
	Maximum	29	266	707	707	707	93	2735
	Mean	0.88	3.42	32.49	29.36	7.06	25.77	171.02
	Median	**0**	**0**	21	18	1	17	47
	Std. Dev.	2.53	18.51	36.14	34.96	29.48	23.95	286.85

Appendix C: Correlation coefficient between cohesion and coupling

Cohesion metrics	System	NOC	NOC*	CBO	CBO_ NA	CBO_ UIB	CBO_ U	RFC
LCC	*ET++*	-0.10	-0.05	-0.11	-0.10	0.04	-0.23	-0.05
	System-B	-0.06	-0.08	-0.02	-0.01	-0.05	-0.03	-0.07
LCOM	*XForms*	0.12	-0.01	0.06	0.11	0.17	-0.17	0.33
	ET++	0.30	0.31	0.44	0.45	0.39	0.21	0.38
	System-B	0.08	0.21	0.28	0.30	0.32	0.07	0.36

Appendix D: Impacts results for the three test systems

	Change	System	Min.	Max.	Mean	Median	Std. Dev.
1.	Variable type change	*XForms*	1	20	**1.78**	1	3.17
		ET++	1	81	**2.02**	1	5.97
		System-B	1	32	**1.46**	1	1.85
2.	Variable scope change from public to protected	*XForms*	-	-	**-**	-	-
		ET++	0	80	**3.78**	0.67[2]	12
		System-B	0	52	**1.84**	1	6.21
3.	Method signature change	*XForms*	1	3.67	**1.19**	1	0.49
		ET++	1	17.64	**1.46**	1	1.26
		System-B	1	38.60	**1.77**	1	2.06
4.	Method scope change from public to protected	*XForms*	0	2.67	**0.18**	0	0.48
		ET++	0	16.64	**0.40**	0	1.19
		System-B	0	37.39	**0.60**	0	1.79
5.	Class derivation change from public to protected	*XForms*	0	4	**0.32**	0	0.89
		ET++	0	281	**3.71**	0	16.46
		System-B	0	291	**4.42**	0	20.03
6.	Addition of abstract class in class inheritance structure	*XForms*	1	64	**4.90**	**1**	11.49
		ET++	1	393	**8.84**	**2**	31.16
		System-B	1	743	**11.34**	**2**	38.20

[2] Note that the impact values are calculated as averages (see Section 3.2), and hence a median need not be an integer.

Appendix E: Correlation coefficient between cohesion and impact

	Change	System	LCC	LCOM
1.	Variable type change	*XForms*	-0.52	0.11
		ET++	0.11	0.18
		System-B	-0.06	0.02
2.	Variable scope change from public to protected	*XForms*	[3]	[1]
		ET++	0.31	0.06
		System-B	-0.10	-0.01
3.	Method signature change	*XForms*	-0.44	-0.01
		ET++	-0.02	0.31
		System-B	-0.07	0.21
4.	Method scope change from public to protected	*XForms*	-0.44	0.09
		ET++	0.25	0.12
		System-B	0.01	0.13
5.	Class derivation change from public to protected	*XForms*	-0.52	0.30
		ET++	0.01	0.36
		System-B	-0.01	0.05
6.	Addition of abstract class in class inheritance structure	*Xforms*	-0.39	-0.01
		ET++	0.03	0.34
		System-B	-0.07	0.35

[3] There is one class in the test set.