Class Cohesion Revisited: An Empirical Study on Industrial Systems

Hind Kabaili, Rudolf K. Keller, François Lustman and Guy Saint-Denis
Département IRO
Université de Montréal
C.P. 6128, succursale Centre-ville
Montréal, Québec H3C 3J7, Canada
E-mail: {kabaili | keller | lustman | stdenisg}@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 the realm of object-oriented systems, experiments have been conducted showing that coupling between classes is such an indicator. However, class cohesion has not been quantitatively studied in respect to changeability. In this research, we set out to investigate whether low cohesion is correlated to high coupling and thus is a changeability indicator, too. As cohesion metrics, LCC and LCOM were adopted, and for measuring coupling, the Chidamber and Kemerer coupling metrics and variants thereof were used. The data collected from three test systems of industrial size indicate no such correlation. Suspecting that the cohesion metrics adopted for the experiment do not adequately capture the cohesion property, we analyzed manually the classes with lowest cohesion values. We found various reasons why these classes, despite of their low cohesion values, should not be broken into smaller classes. We conclude that cohesion metrics should not only be based on common attribute usage between methods and on method invocation, but also on patterns of interaction between class members and, ultimately, on the functionality of methods and attributes.

Keywords: changeability, design metrics, object-oriented, cohesion, coupling, empirical validation.

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).

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 software. Industrial buyers want to be sure of the product quality 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, is interested in buying large-scale software rather than developing it. It needs to be sure of the quality of the systems it acquires. The *SPOOL* project *(Spreading desirable Properties into the design of Object-Oriented, Large-scale software systems*), is a joint industry/university research project between the *Quality Engineering and Research* team of *Bell Canada* and the *GELO* group at the *Université de Montréal*. As part of the project, design properties are investigated as changeability indicators.

Cohesion is an important quality property of OO designs. Several metrics have been proposed to quantify and measure this design property. In this paper, we try to assess cohesion as an indicator of changeability. In some previous works, coupling has been validated as a quality indicator. By showing a correlation between cohesion and coupling, we will be able to assert that cohesion is quality indicator, too. The paper is organized as follows. Section 2 presents an overview of cohesion as a quality indicator and describes a potential relationship between cohesion and coupling. This relationship was tested empirically, as reported in Section 3. The negative result of the test led us to investigate the reasons behind this lack of relationship. This investigation is described in Section 4. Section 5, finally, summarizes the work and provides an outlook into future work.

2 Cohesion and design quality

Building quality OO systems relies on good design. To assess with some objectivity the quality of a design, we need to quantify design properties. Several software metrics have been developed to assess and control the design phase and its products. One of the most important criteria in OO design is cohesion. Module cohesion was introduced by Yourdon and Constantine as "how tightly bound or related the internal elements of a module are to one another" [YC79]. 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.

2.1 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) [CDK94]. 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 sharing at least one attribute. The metric is set to zero when the value is negative.

Li and Henry [LH93] redefine LCOM as the number of disjoint sets of methods accessing similar instance variables.

Hitz and Montazeri [HM95] 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 [BK95] 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. 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. 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, as the prime cohesion metrics of our experimentation.

2.2 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. Class coupling is usually defined as class interaction.

Many metrics that capture interactions between classes have been defined. Chidamber and Kemerer proposed two coupling metrics [CDK94] that have been validated as fault prone indicators [BBM96]:

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 [BDM97].

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 specifics quality aspect. We decided to investigate the potential of cohesion metrics as changeability indicators by looking for relationships between cohesion and coupling.

3 Empirical validation of cohesion-coupling relationship

3.1 Objectives

Most large-scale software systems have a long life span. Over the years, they require changes to improve performance, to address new needs, or to adapt the system to a changing environment. Since our industrial partner has a vested interest in software changeability, we conducted our experiment with respect to changeability. One way to assess the changeability of a software system is to find some design properties that can be used as changeability indicators.

In the realm of OO systems, experiments have been conducted showing that coupling between classes is an indicator of changeability. Chaumun et al. defined a model of software changes and change impacts at the conceptual level. They observed a high correlation between changeability and some coupling metrics, across different industrial systems and across various types of changes [CKKL99].

However, class cohesion has not been studied quantitatively with respect to changeability. Weak class cohesion leads to high coupling with the rest of the system, and thus to high change impact. Weak class cohesion is therefore expected to result in poor changeability. One way to investigate cohesion as a changeability indicator, is to prove whether low cohesion is indeed correlated to high coupling.

Such a proof would confirm our intuition that there is a correlation between cohesion and changeability. We are aware that this latter hypothesis would require a study in its own right, which is beyond the scope of this paper.

3.2 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 [BBM96] and changeability [CKKL99, CKK+99]. 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)" [CDK98]. Below, we present the four metrics, together with the rationale for their consideration.

NOC (Number Of Children in subtree):* 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.

3.3 Experimentation set up

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.

Three industrial systems were considered. They vary in class size and application domain. The first test system is *XForms*, which can be freely downloaded from the web [Xfo97]. 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 [WGM98]. The version used in the experiment is the one included in the SNiFF+ development environment [Tak99]. 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. Note that header files from the compiler are included in the numbers shown in the lower part of the table (last six rows), whereas the numbers in the upper part (first four rows) represent the system that was effectively investigated in the study.

Table 1: Size metrics of test 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 classes	221	722	1 420
# of files (.C/.h)	143	485	1 153
# of generalizations	75	466	941
# 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

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 [KSRP99].

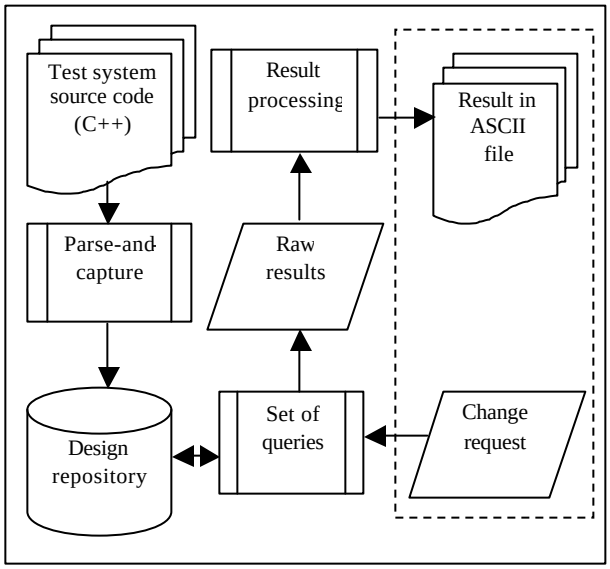

Figure 1: Environment for metrics calculation

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* [Dev92], 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 [RJB99]. The OO database management system *POET 5.1* [Poe99] 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.

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 tables 2 and 3). To test the four correlation hypotheses, we calculated for the two cohesion metrics the Pearson coefficient of correlation in respect to the seven metrics of Section 3.2 (see Appendix C).

3.4 Results

Descriptive statistics of the three test systems are summarized in Appendix A. 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.

Appendix B shows descriptive statistics of the cohesion metrics of the three test systems. Based on these values, and referring to the definition of both LCOM and LCC, we early concluded that the three test systems are not strongly cohesive.

The Pearson correlation coefficients are presented in Appendix C. We found no correlation 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 seven 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.

4 Class cohesion and methods coupling

The goal of our study was to find a correlation between cohesion and coupling, but the result was negative. Consequently, we set out to reason about this absence of correlation, especially with respect to the two genuine coupling metrics CBO and RFC.

4.1 Reasoning about results

Given negative results reported above, we came up with the following two explanations:

(1) The cohesion metrics or the coupling metrics chosen for the experimentation are not the right ones.

(2) There is no relationship between cohesion and coupling whatsoever.

Hypothesis (2), being counter to a widely-held belief in the design community, was discarded. We then focused our investigation on hypothesis (1).

From hypothesis (1) we derived the following two sub-hypotheses:

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

(1B) CBO and RFC do not measure the coupling property of a class as stated by Yourdon and Constantine.

Sub-hypothesis (1B) was rejected on the grounds that the coupling metrics adopted in the study are quite well understood and validated. Basili et al., for instance, validated CBO and RFC as quality indicators [BBM96]. Another example of validation is the work of Chaumun et al., who experimented with extended C&K design metrics as changeability indicators [CKK+99].

On the other hand, we question 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 are 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.

4.2 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 classes that model 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 consolidate 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. Appendix D lists, for illustration, some of these classes. We came up with four major reasons for not splitting them.

First, some classes had no variables (such as the class *Glob* in *ET++*; see Appendix D) or only abstract methods (such as class *App* in *XForms*), 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. The *PeCollectClients* class in *ET++* is such an example. It contains three methods sharing the same instance variables. LCC should be equal to 1, but inherited methods reduce LCC to 0.3 (see Appendix D). Note that in this and several other examples, the three ways of calculating LCC led to widely different cohesion results.

Third, some classes have multiple methods that share no variables but perform related functionality. Consider for example the class *RevObjListIter* in *ET++* (Appendix D) which is used to manage lists of objects. Its methods carry out different list management operations on lists that are passed to the class as parameters. 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. For example, the class *DRRequests* in *System-B* contains 22 attributes together with pairs of *get* and *set* methods for each of them. However, these attributes belong together and should not be separated.

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.

4.3 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 metrics will have to go beyond the simple sharing of class variables and capture additional information.

Briand et al. provide a categorization of cohesion metrics [BDPW98]. 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 [CK98]. 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.

5 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.

According to OO design principles, a good design exhibiting high class cohesion goes together with low coupling between classes. A relationship should therefore exist between cohesion and coupling. 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.

6 Reference

[BBM96] Victor R. Basili, Lionel C. Briand, and Walcelio L. Melo. A validation of object-oriented design metrics as quality indicators. In *IEEE Transactions on Software Engineering*, 22(10): 751-761, October 1996.

[BDM97] Lionel Briand, Prem Devanbu, and Walcelio Melo. An investigation into coupling measures for C++. In *Proceedings of the International Conference on Software*

Engineering (ICSE'97), pages 412-421, Boston, MA, May 1997.

[BDPW98] Lionel C. Briand, John Daly, and Jurgen Wust. A unified framework for cohesion measurement in object-oriented systems. In *Empirical Software Engineering - An International Journal*, 3(1), pages 67-117, 1998.

[BK95] James M. Bieman and Byung-Kyoo Kang. Cohesion and reuse in an object-oriented system. In *Proceedings of the Symposium on Software Reusability (SSR'95)*, pages 259-262, Seattle, WA, April 1995.

[CKKL99] M. Ajmal Chaumun, Hind Kabaili, Rudolf K. Keller, and Francois Lustman. A change impact model for changeability assessment in object-oriented systems. In *Proceedings of the Third Euromicro Working Conference on Software Maintenance and Reengineering*, pages 130-138, Amsterdam, The Netherlands, March 1999.

[CKK+99] M. Ajmal Chaumun, Hind Kabaili, Rudolf K. Keller, Francois Lustman, and Guy St-Denis. Design properties and object-oriented software changeability. Technical Report GELO-92, Universite de Montreal, Montreal, Quebec, Canada, March 1999. Submitted for publication.

[CDK94]S. R. Chidamber and C. F. Kemerer. A Metrics Suite for Object Oriented Design. In *IEEE Transactions on Software Engineering*, Vol. 20, No. 6, pages 476-493, June 1994.

[CDK98] Shyam R. Chidamber, David P. Darcy, and Chris F. Kemerer. Managerial use of metrics for object-oriented software: An exploratory analysis. In *IEEE Transactions on Software Engineering*, 24(8):629-639, August 1998.

[CK98] Heung Seok Chae and Yong Rae Kwon. A cohesion measure for classes in object-oriented systems, In *Proceedings of the Fifth international Software Metrics Symposium*, pages 158-166, Bethesda, MD, November 1998.

[Dev92] Premkumar T. Devanbu. GENOA - a customizable, language- and front-end independent code analyzer, In *Proceedings of the 14th International Conference on Software Engineering (ICSE'92*, pages 307-317, Melbourne, Australia, 1992.

[HM95] Martin Hitz and Behzad Montazeri. Measuring coupling and cohesion in object-oriented systems. *Proc. Int. Symposium on Applied Corporate Computing*, pages 25-27, October, 1995.

[KSRP99] Rudolf K. Keller, Reinhard Schauer, Sebastien Robitaille, and Patrick Page. Pattern-based reverse engineering of design components. In Proceedings of the Twenty-First International Conference on Software Engineering, Los Angeles, CA, May 1999. IEEE. to appear.

[LH93] Wei Li and Sallie Henry. Object-oriented metrics that predict maintainability. In *Journal of Systems and Software*, 23:111-122, February, 1993.

[Poe99] Poet Software Corporation, San Mateo, CA. POET Java ODMG Binding. Online documentation, 1999. Available online at <http://www.poet.com/>.

[RJB99] James Rumbaugh, Ivar Jacobson, and Grady Booch. The Unified Modeling Language Reference Manual. Addison-Wesley, 1999.

[Tak99] TakeFive GmbH, Salzburg, Austria. SNiFF+ Documentation Set, 1999. Available online at: <http://www.takefive.com>.

[WGM89] Andre Weinand, Erich Gamma, and Rudolf Marty. Design and implementation of ET++, a seamless object-oriented application framework. In *Structured Programming*, 10(2): 63-87, April-June, 1989.

[Xfo97] Xforms Library. Graphical user interface for X. Documentation Set, 1997.

Available online at
<http://bragg.phys.uwm.edu/xforms>.

[YC79] Edward Yourdon and Larry L. Constantine. Structured Design. Prentice Hall, Englewood Cliffs, N.J., 1979.

Appendix A: Descriptive statistics of the three test systems

System		NOC	NOC*	CBO	CBO_NA	CBO_IUB	CBO_U	RFC
XForms	**Minimum**	0	0	0	0	0	0	0
83 classes	**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++	**Minimum**	0	0	0	0	0	0	0
584 classes	**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	**Minimum**	0	0	0	0	0	0	0
1226 classes	**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 B: Metrics results for the three test systems

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

Appendix C: Correlation coefficients for the three test systems

Cohesion Metrics	System	NOC	NOC*	CBO	CBO_NA	CBO_UIB	CBO_U	RFC
LCC	*XForms*	-0.09	-0.15	-0.17	-0.10	-0.030	-0.22	-0.17
	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: Example of weakly cohesive class that are not candidate for splitting

Classes	Systems	LCC	LCOM
Glob	*ET++*	0	3
PeCollectClients	*ET++*	0.3	0
App	*XForms*	0	6
RevObjListIter	*ET++*	0.16	1
DRRequests	*System-B*	0.023	858