Coupling and Cohesion as Modularization Drivers: Are we being over-persuaded?

Fernando Brito e Abreu
FCT/UNL and INESC
Software Engineering Group
R. Alves Redol, 9
1000 Lisboa, Portugal
+351 213100263
fba@di.fct.unl.pt

Miguel Goulão
FCT/UNL and INESC
Software Engineering Group
R. Alves Redol, 9
1000 Lisboa, Portugal
+351 213100263
miguel.goulao@di.fct.unl.pt

ABSTRACT
For around three decades Software Engineering gurus have "sold" us the ideal of minimal coupling and maximal cohesion at all levels of abstraction as a way to reduce the effort to understand and maintain software systems. The object-oriented paradigm brought a new design philosophy and encapsulation mechanisms that apparently would help us to achieve that desideratum. However, after a decade where this paradigm has emerged as the dominant one, we are faced with practitioners' reality: coupling and cohesion do not seem to be the dominant driving forces when it comes to modularization. This conclusion was based on a relatively large sample of heterogeneous systems. We describe an environment that allows not only assessing this reality but also deriving better modularization solutions in what concerns coupling and cohesion. These solutions are generated by means of cluster analysis techniques and partially preserve the original modularization criteria. We believe this approach can be of great help in reengineering actions of object-oriented legacy systems.

Keywords
Coupling, Cohesion, Modularity, Clustering Analysis

1 MOTIVATION

According to the software quality model proposed in the ISO9126 standard, modularity is an internal quality attribute influencing the external software quality characteristics [1]. Modularity should be, and usually is, applied at all levels of abstraction, ranging from requirements specifications to the executable code level. It can be considered a fundamental engineering principle since it allows, among other things:

- to design and develop different parts of the same system by different people, often belonging to distinct companies;
- to handle the complexity of large systems by splitting loosely coupled parts that can be better understood individually;
- to test systems in a parallel fashion (different people simultaneously);
- to substitute or repair defective parts of a system without interfering with other parts;
- to reuse existing parts in different contexts;
- to divide the system in configuration units to be put under configuration control;
- to restrict change propagation.

In this paper we will analyze the design modularity of OO systems. At this level of abstraction a module is as set of classes. We will use information on the dependencies among classes to assess modularity. Software engineering gurus have stressed, *ad nausea*, the importance of high cohesion and low coupling [2-6] as the main drivers to module identification. Here, the coupling among classes within the same module (intra-modular class coupling) traduces **module cohesion** and the coupling among classes belonging to different modules (inter-modular class coupling) corresponds to **module coupling**.
Modules as groups of related classes are named heterogeneously in the literature of the object-oriented paradigm. C++ has **namespaces** [7], Delphi has **units** [8], while Java [9], SmallTalk [10], UML [14] and Catalysis [11] use **packages**. Bertrand Meyer's cluster model uses **clusters** [12], both OMT [13] and Objectory [3] considers **sub-systems**, the Booch Method [15] has **categories,** while Meilir Page-Jones mentions **domains** and **sub-domains** [16]. No matter the designation used, all authors recognize the need to aggregate classes into clusters. Albeit this recognition, most of the heuristics provided on modularization for object-oriented systems are of a qualitative nature or, at best, provide some vague quantitative guidelines for clustering classes.
In this paper we define a **quantitative approach** to assess modularization and derive improved alternatives based on class coupling and cohesion. A tool named MOTTO (MOdularization Trial Tool for Object oriented systems), which is briefly described, was built to support this research. Based on a relatively large sample, we will show that, very often, the criteria used by practitioners to modularize object-oriented systems falls short to achieve the objective of minimal coupling and maximal cohesion.

This paper is organized as follows: in section 2, the underlying concepts and techniques of our quantitative approach to the modularization of OO systems are described. Section 3 includes some details on the tool that has been developed to support this research. Section 4 contains a description of the methodological approach followed, including our working hypotheses, the experiment itself and a critical analysis on the validity of the approach. In section 5 data collected during our experiments is analyzed, to test the previously formulated hypotheses. Conclusions are drawn in section 6 and related and future works are identified in section 7.

2 A QUANTITATIVE APPROACH

2.1 The quest for an optimal modularization

References on modularization for OO systems usually provide some qualitative criteria to help achieve the best modularization configurations. Among these rules-of-thumb or heuristics the most frequent one is: *"it is desirable that modules are highly cohesive and loosely coupled"*. This may be valuable in the process of understanding the sort of class clusters we should strive for, but is clearly insufficient when it comes to automating the clustering process. How much is meant by "highly" and by "loosely"?

A quantitative criterion is thus required to deal with this problem. A few questions arise when one tries to define it:

1. Given a system, how much should one decompose it in modules (i.e., how many modules should one define)?
2. How disparate in size should the modules in that system be?
3. How should the classes be distributed amongst the modules?

The brief quantitative references on this subject lack a proper validation. Jacobson mentions 1 to 5 classes as a good size for modules, although he admits having used as much as 17 [3, pp.145]. Meyer advocates that systems should have modules of 5 to 40 classes [12]. The definition of this sort of threshold-based rules-of-thumb, apparently tempting, implies some uniformity in module dimension (number of classes per module). Forcing designers into that uniformity would be an artificial imposition that would result in ill-defined modularization solutions. Nevertheless we believe that dispersion on module size should be somehow constrained to avoid extremely skewed distributions (e.g. N-1 modules with just 1 class and all the remaining classes in a Nth module). As a relative dispersion statistic we used ***RMD*** *(Relative Module Dispersion)*, defined as follows:

$$RMD = \frac{CM_{MAX} - CM_{MIN}}{AMM}$$

where CM_{MAX} and CM_{MIN} represent, respectively, the maximum and minimum number of classes in the current set of modules and ***AMM*** *(Average Module Membership)* is defined as:

$$AMM = \frac{NC}{NM}$$

where ***NC*** the number of classes and ***NM*** is the number of modules. Notice that, as expected, the dispersion is null (*RMD=0*) when all modules have the same number of classes.

Now, picking the word of the Software Engineering gurus, we would like to maximize module cohesion (intra-modular class coupling), while minimizing module coupling (inter-modular class coupling). This desideratum can be expressed by the maximization of the ***ICD*** *(Intra-modular Coupling Density)* metric, defined as follows:

$$ICD = \frac{CI_{IN}}{CI_{IN} + CI_{OUT}}$$

where CI_{IN} is the number of class couplings (Coupling Instances) within modules and CI_{OUT} the number of couplings between classes belonging to distinct modules. $CI_{IN} + CI_{OUT}$ corresponds to the total number of existing class couplings. The solution that has the higher ***ICD*** is the one that maximizes cohesion within modules while minimizing the coupling between them. However, it is not hard to realize that that solution is the one where **all classes reside in just one module**! This is clearly not our goal, as no system partitioning is achieved. In the other extreme we have one class per module (i.e. as many modules as there are classes). That is also opposing the ideal of grouping underlying modularization itself. Then, we are only concerned with ***NC-2*** different number of modules, corresponding to the interval $NM \in [2, NC-1]$.

On a previous paper we proposed the use of **cluster analysis** to guide the reengineering of the modularity of OO systems [17]. Cluster analysis is a set of techniques concerned with the classification of similar items into groups (clusters). These techniques require a measure of the similarity among classes. Here we use the coupling level between each pair of classes as a measure of their **similarity**, or **affinity**. In this sense, the degree of similarity reflects the strength of the coupling between the two classes. To be more specific, the affinity between two classes is a monotonous increasing function of their number of coupling relations. The rationale is that classes that are strongly coupled should belong to the same cluster (hereby called module), while loosely coupled classes should be placed in distinct modules.

Our input to calculate the affinity for a given system is a list of coupling instances, each characterized by the tuple:

Coupling instance :
(origin class, destination class, coupling category)

Twelve coupling categories are used in our taxonomy (upper frame in *Figure 1*). Since our research is exploratory in nature, we have tried five different ways of combining the information contained in the coupling instances to derive the affinity (lower frame in *Figure 1*). The weights used in some of these rating schemes (the "weighted" ones, obviously) vary in the range [1, 10] and are calculated with information extracted from the original solution. Each weight for a given coupling category is given by:

$$\boldsymbol{a}_{CC} = \begin{cases} CI_{CCOUT} = 0: & 10 \\ CI_{CCIN}, CI_{CCOUT} = 0: & 1 \\ else: \quad Round(0.5 + 10 \times \dfrac{CI_{CCIN}}{CI_{CCIN} + CI_{CCOUT}}) & \end{cases}$$

where $\boldsymbol{CI_{CCIN}}$ and $\boldsymbol{CI_{CCOUT}}$ are, respectively, the number of intra-modular and inter-modular coupling instances for that category in the original modularization proposal. This technique partially preserves the original modularization criteria, at least as their developers perceived it.

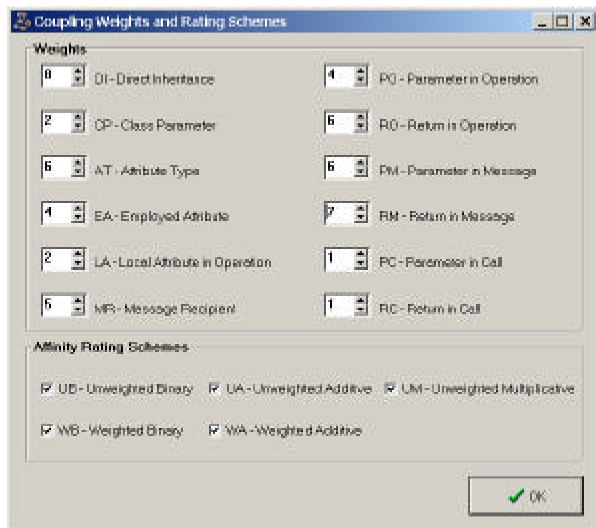

Figure 1 - A configuration window from MOTTO

The explanation of the coupling categories, the mathematical definition of the affinity rating schemes and the description of the transformation applied to the affinities to build a **dissimilarities matrix** (where each item corresponds to a pair of classes), can be found in [17]. This matrix is the basic input for the clustering algorithm.

Assuming that we know how many modules we wish to create, cluster analysis provides us a distribution of classes among modules that maximizes the affinity of classes within each module, while minimizing the affinity of classes from different modules. A number of distinct clustering methods exists. We have used seven well-known hierarchical agglomerative clustering methods (left frame in *Figure 2*). Agglomerative methods start with all items in separate clusters and proceed in iterations joining them until the defined number of clusters is reached [18].

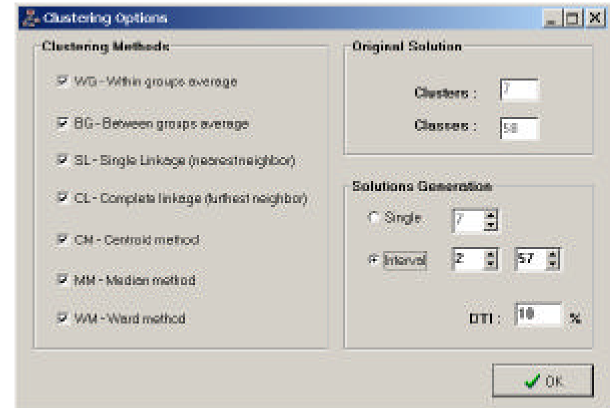

Figure 2 – Other configuration window from MOTTO

With five affinity-rating schemes and seven clustering methods, we obtain ***35*** distinct modularization solutions for a fixed number of modules. We thus have to define a figure of merit to find out which is the best solution for modularization among the ***(NC-2)*35*** generated solutions. We can divide this problem in two: finding the best solution for a fixed number of modules among the 35 alternatives, and then comparing the best solutions found for each of the ***NC-2*** module decomposition alternatives to derive the best of the best. The criteria chosen for each of these two steps is different as will now see.

2.2 A figure of merit for modularization

We previously observed that *Intra-modular Coupling Density* (***ICD***) cannot be used, **alone,** as a modularization metric. However, for constant ***NM*** and ***NC***, the modularization will be better if ***ICD*** increases. On the other hand, for constant ***NC*** and ***ICD***, the modularization will be better if the number of modules (***NM***) increases. In other words, we should split a given system in modules as much as possible as long as ***ICD*** is not sacrificed (reduced). Still, for constant ***NM*** and ***ICD***, the modularization is worst when the number of classes increases, since we would then be increasing the average module size (***AMM***) without increasing ***ICD***.
This line of thought leads us naturally to the formulation of the *Modularization Merit Factor* (***MMF***):

$$MMF = ICD^{SDI} \times \frac{NM}{NC} = \frac{ICD^{SDI}}{AMM}$$

where ***SDI*** is the *System Defragmentation Index*, a calibrating factor of our approach. The value of ***SDI*** allows the analyst to increase or decrease the relative weight of ***ICD*** in the computation of ***MMF***. Lower values of ***SDI*** will foster solutions with more clusters, while higher values of will favor less fragmented solutions. In the experiment described in this paper we have used 2 as the value for ***SDI***.

2.3 The best solution for a fixed number of modules

As defined, ***MMF*** may not lead to the best modularization solution since its definition does not take into account the dispersion on the module sizes. The proposed solution may be highly skewed, as a result of the maximization of ***ICD***. A large number of classes would concentrate on a small set of modules, while the remaining ones would be scattered over the remaining modules (typically one class per each). To overcome this problem we have set an upper limit to the dispersion:

$$RMD_{MAX} = RMD_{OR} \times (1 + DTI)$$

where $\boldsymbol{RMD_{OR}}$ is the relative module dispersion of the original clustering solution and ***DTI*** is the *Dispersion Tolerance Index* (lower-right frame in *Figure 2*). The best solution is then the one that maximizes ***MMF***, chosen amongst the ones that fulfill the previous dispersion criteria.

DTI represents our willingness to maintain the level of relative module dispersion close to the original one. Lower values of ***DTI*** will provide solutions with a more similar ***RMD*** to the original solution. In this experiment, we have used a ***DTI*** value of 10%.

Considering both ***RMD*** and ***MMF*** presents a multiple-criteria decision problem. Multiple-criteria decisions encompass the consideration of often conflicting, objective functions. In this case, we have limited the solution space based on one criterion (low dispersion) and then applied the other criterion (***MMF*** maximization). The chosen solution is then the one that has a high ***MMF*** and not too much dispersion.

2.4 The best overall

Determining the best overall solution implies choosing a number of clusters in the admissible interval. The algorithm here is much simpler than in the first step since the solutions derived then already keep the class dispersion in modules within an acceptable range. The best solution is then, among the set of ***NC-2*** solutions derived in the previous step, the one that maximizes ***MMF***.

3 TOOL SUPPORT

We have developed the MOTTO tool to compute the best coupling solutions according to the rationale depicted in section 2. MOTTO is used in conjunction with MOODKIT G2 and SPSS. SPSS (Statistical Package for Social Sciences) is a commercial statistics package tool. MOODKIT G2 is the second generation of a tool originally conceived for the collection and analysis of OO design metrics [19]. It generates the categorized coupling instances from an OO design expressed in GOODLY [20].

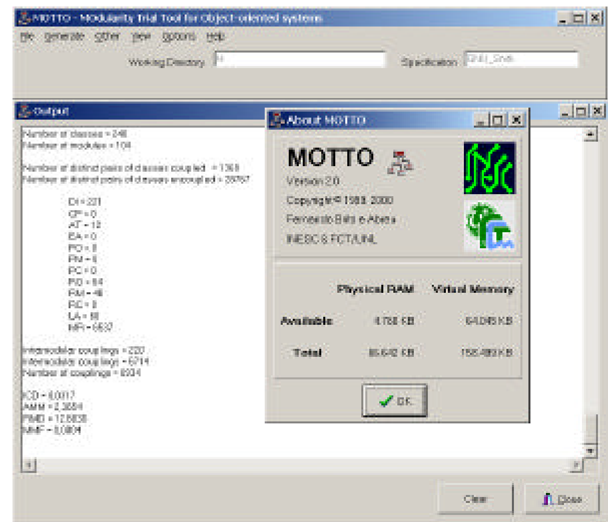

Figure 3 - The MOTTO tool

GOODLY (a Generic Object-Oriented Design Language? Yes!) can be derived from OO design models expressed in several notations (UML, Booch and OMT) or from some OO programming languages (C++, Eiffel, Smalltalk). The list of coupling instances is fed into MOTTO that, in turn, generates a couplings table. From the latter it computes a dissimilarities matrix, a symmetric squared matrix, where the rows and columns represent existing classes and the values represent the dissimilarity between each pair of them. MOTTO also produces an appropriate SPSS script to drive the clustering process. This script is then executed by the SPSS processor, which generates a set of clustering solutions. MOTTO takes over the process once more to compute two-dimensional ***MMF*** matrices, one for each considered number of modules. The dimensions of these matrices are the clustering method and the rating scheme. From these matrices MOTTO chooses the best solution, according to the criteria described in the previous section. *Figure 4* outlines the data flows of this process. The sequence of events is presented in *Figure 5* using UML sequence diagrams [14].

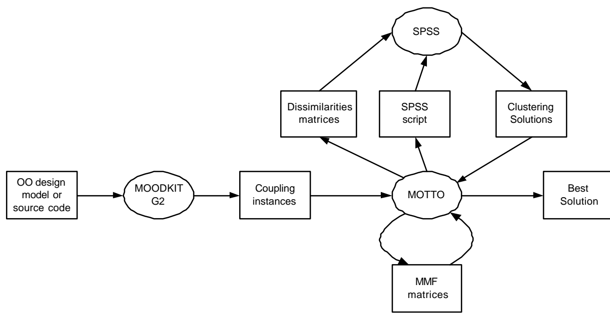

Figure 4 – Clustering process flows.

The usage of MOTTO is relatively straightforward. The tool includes both the possibility of stepwise and quick (batch) processing. In the former case we may use the available viewing facilities to verify all the intermediate results it generates. The quick generation option allows the user to execute the whole process described earlier in this section with a single click.
MOTTO exports its data files in tab-delimited format that can be easily imported by spreadsheets, statistics packages or databases.

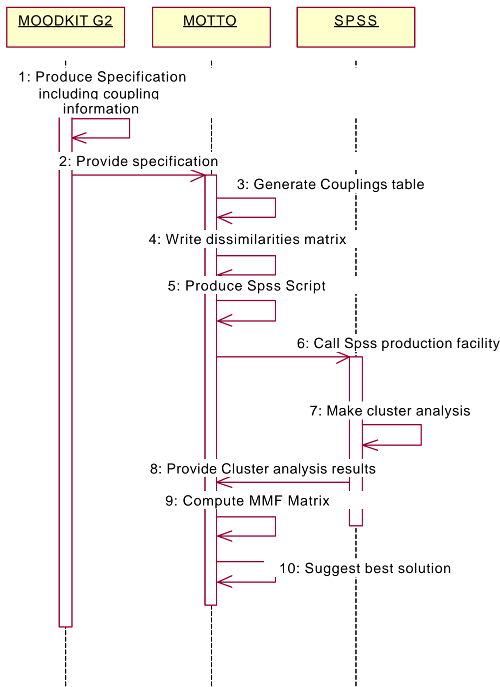

Figure 5 – Sequence diagram of the clustering process

We plan to make this tool freely available for research purposes. Please contact us in case you might consider using it.

4 METHODOLOGICAL APPROACH

4.1 Experiment definition

In this experiment we want to assess if practitioners use coupling and cohesion as the dominant driving forces in the modularization of OO systems. We also want to test the feasibility of an automated approach to derive improved modularization solutions on what concerns the same design aspects.
Our experiment can be defined using a goal definition format [Wohlin, 2000 #4703], as follows:

- **Object of study**: OO systems modularization.
- **Purpose of study**: evaluate the influence of coupling and cohesion in modularization criteria and show how systems modularization can improved regarding these two design aspects.
- **Quality focus and perspective of the study**: Systems maintainability, from the structural point of view.
- **Context**: The experiment is conducted in a research environment, and its subjects (sample items) come from both academia and industry.

4.2 Research Questions

The majority of our research questions are set to identify which factors influence the improvement achieved in the modularity. Naturally, we start by assessing the modularization improvements achieved with our technique.

a) Are we really improving modularization?
If we find out that the proposed modularization solutions are consistently better (at least in what concerns coupling and cohesion) than the ones that are implemented in the analyzed systems, we can conclude that our approach leads to better solutions

b) Does size matters?
It is important to find out whether or not the effectiveness of our technique is immune to systems size variation.

c) Are some clustering methods better than others?
If some clustering methods consistently provide better modularization solutions than others, we can speed-up the solution computation by only using the most effective method(s).

d) Are some affinity weighting schemes (used for calculating the dissimilarity matrices) better than others?
The reasoning here is similar to the one on research question b). It is possible to speed-up the computational process by selecting only the most effective weighting scheme(s).

e) Are we proposing a different level of partitioning?

If we find out that the proposed solutions systematically partition the system into a larger number of modules than the original, then this may indicate that practitioners tend to group classes that are not significantly related into the same module.

4.3 Sample Selection

A sample of 23 object-oriented systems, produced both in academic and industrial settings, is used in this experiment. It includes systems specified in Delphi, Eiffel, Smalltalk and UML. These are disparate in size, with systems ranging from 8 to over 250 classes. There is also a clear diversity on the application domain. The sample includes, among other things, programming environments, class libraries and applications. Appendix I includes some details for the items in our sample.

5 DATA ANALYSIS

5.1 Hypotheses Formulation

For each of our research questions, we formulate a research hypothesis and its null counterpart, which is then tested. If the null hypothesis is rejected, we can accept our research hypothesis.

5.2 Data considerations

Some data analysis is required so that we can choose appropriate statistical tests.

We start by analyzing the normality of the MMF improvement. We define MMF improvement as:

$$MMF_{IMPROV} = \frac{MMF_{PROP} - MMF_{OR}}{MMF_{OR}}$$

where:

- MMF_{IMPROV} – relative **improv**ement of MMF
- MMF_{PROP} – MMF for the **prop**osed solution
- MMF_{OR} _ MMF for the **or**iginal solution

Table 1 presents some descriptive statistics on the relative improvement distribution. Although the original sample contains 23 systems, 3 of them were removed in the computation of these descriptive statistics, as they presented extreme values (due to an almost null modularity in the original system design, the denominator of the MMF_{IMPROV} formula becomes approximately 0) that would lead to erroneous conclusions. Extreme values were identified by means of a boxplot for MMF_{IMPROV}. Notice that the mean improvement is of 477%, and the lowest improvement was of 37%.

N	Mean	Std.dev.	Min.	Max.	Percentiles		
					25th	50th	75th
20	4.77	5.76	.37	20.26	1.41	2.11	6.58

Table 1 – Descriptive Statistics for MMF_{IMPROV}.

The normality test can be formulated as:

***H_{x0}:** The relative improvement of the solution proposed by our approach has a normal distribution.*
***H_{x1}:** The relative improvement of the solution proposed by our approach does not have a normal distribution.*

We can test the null hypothesis by using a One Sample Kolmogorov-Smirnov test (***Table 2***) for normality. The "Kolmogorov-Smirnov Z" is the absolute difference transformed into a standardized score. The 2-tailed asymptotic significance level represents the probability of the Z value being of that magnitude if the null hypothesis were true. As we are conducting our test at $\alpha = .05$, we cannot reject the null hypothesis since our P value (0.11) is more than our α criterion value. Therefore, the test shows that MMF_{IMPROV} has a normal distribution.

		MMF_{IMPROV}
N		20
Normal Parameters	Mean	4.77
	Std. Dev.	5.76
Most Extreme Differences	Absolute	.27
	Positive	.27
	Negative	-.22
Kolmogorov-Smirnov Z		1.21
Asymp. Sig. (2-tailed) P		.11

Table 2 – Kolmogorov-Smirnov test for normality of MMF_{IMPROV}

One of the research questions concerns the number of modules in the original solutions and their correspondent values in the proposed ones. Both distributions are checked for normality here, by testing the following hypotheses:

***H_{y0}:** The number of modules of the original solution has a normal distribution.*
***H_{y1}:** The relative improvement of the original solution does not have a normal distribution.*

***H_{z0}:** The number of modules of the solution proposed by our approach has a normal distribution.*
***H_{z1}:** The number of modules of the solution proposed by our approach does not have a normal distribution.*

Some descriptive statistics on the number of modules in the original solution (NM_{OR}) and in the improved solution (NM_{IMPROV}) are presented in ***Table 3***.

	N	Mean	Std. Dev.	Min	Max	Percentiles		
						25th	50th	75th
NM_{OR}	23	12.70	22.68	1	104	3	5	11
NM_{Improv}	23	36.04	38.79	2	128	7	22	58

Table 3 – Descriptive Statistics for NM_{OR} and NM_{IMPROV}.

Please notice that the 3 systems that were ruled out for their extreme values in what concerned relative improvement can be included here, since that value is not relevant for this test.
The normality assumption is tested in ***Table 4***.

		NM_{OR}	NM_{IMPROP}
N		23	23
Normal Parameters	Mean	12.69	36.04
	Std. Dev.	22.68	38.79
Most Extreme Differences	Absolute	.36	.29
	Positive	.36	.29
	Negative	-.33	-.19
Kolmogorov-Smirnov Z		1.71	1.38
Asymp. Sig. (2-tailed) P		.01	.04

Table 4 – Kolmogorov-Smirnov test for normality of the number of modules.

Both P values are lower than our α threshold, so we have to reject the null hypothesis, as there is a very low probability that the differences between both distributions and the expected normal ones have occurred by accident. Therefore, we have to assume that neither NM_{OR} nor NM_{IMPROV} have a normal distribution.

5.3 Answering the research questions

The first question in section 4 was about our approach deriving better modularization solutions than the ones found in the original systems. This can be formulated as:

H_{a0}: *The proposed solutions do not improve MMF when compared with the original one.*
H_{a1}: *The proposed solutions improve MMF when compared with the original.*

The rejection of H_{a0} and consequent acceptance of H_{a1} does not require further statistical testing. It can be seen both in ***Table 1*** and Appendix I that for all the tested systems, we were able to achieve noticeable improvements on MMF.
The lowest MMF_{IMPROV} obtained in our experiment was of 37%. Such an improvement of a system's modularity is still a very interesting one. It occurred on a graphics package developed in Delphi to serve as a basis for a textbook on Object-Oriented Programming in Pascal [21]. This is hardly a surprise: where else can one expect to find examples where special care is put into modularity, if not in a book where among other things the concept itself is taught?

The influence of size in our approach's effectiveness is the basic concern of question *b*. The corresponding hypotheses are formulated here:

H_{b0}: *The relative improvement of the solution proposed by our approach does not depend significantly on the systems size.*

H_{b1}: *The relative improvement of the solution proposed by our approach depends significantly on the systems size.*

To check H_{b1}, we test its null counterpart. The earlier mentioned 3 extreme values for MMF_{IMPROV} remain out of this test, of course. ***Figure 6*** represents a scatterplot of the number of classes and the corresponding MMF_{IMPROV}, where there does not seem to be an association between the two variables. A Pearson correlation test resulted in a correlation coefficient of 0.367 at a significance level of 0.111, for our 20 systems, thus confirming that there is no evidence of an association between the two variables. So, we cannot reject our null hypothesis. The effectiveness of our approach is not significantly affected by systems size.

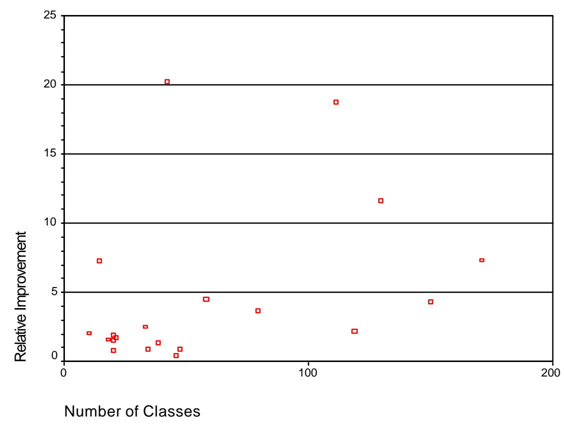

Figure 6 – Scatter plot diagram for classes vs. MMF_{IMPROV}.

Finding out whether or not some clustering methods are more effective than others can lead to substantial speedups in the modularity proposal efficiency (question *c*, in section 4). The following hypotheses can be formulated:

H_{c0}: *The chosen clustering method has no significant influence on the relative improvement of the solution proposed by our approach.*
H_{c1}: *The chosen clustering method has a significant influence on the relative improvement of the solution proposed by our approach.*

Hypothesis H_{b1} concerns the selection of an appropriate clustering method. If all the methods have the same effectiveness level, we expect the selected solutions to be distributed in a relatively equal share. ***Table 5*** shows that this assertion does not hold in our sample. WM and BG alone are responsible for more than 52% of the solutions, while methods such as CL and CM are the less often selected.

If time is a constraint, one may sacrifice the least effective methods, i.e., the ones that provide less effective solutions.

	Frequency	Percent	Cumulative Percent
CL	1	4.3	4.3
CM	1	4.3	8.7
WG	2	8.7	17.4
MM	3	13.0	30.4
SL	4	17.4	47.8
BG	5	21.7	69.6
WM	7	30.4	100.0
Total	23	100.0	

Table 5 – Solutions proposed by each clustering method

A similar study can be performed for weighting schemes. The corresponding hypotheses are formulated as:

***H_{d0}*:** *The weighting schemes have no significant influence on the relative improvement of the solution proposed by our approach.*
***H_{d1}*:** *The weighting schemes have a significant influence on the relative improvement of the solution proposed by our approach.*

Hypothesis H_{d1} concerns the selection of the most appropriate weighting schemes. Again, we use the solutions distribution to assess this hypothesis.

	Frequency	Percent	Cumulative Percent
UB	0	.0	.0
WB	0	.0	.0
UA	1	4.3	4.3
UM	6	26.1	30.4
WA	7	30.4	60.9
WM	9	39.1	100.0
Total	23	100.0	

Table 6 – Solutions proposed by each weighting scheme

The first noticeable fact about ***Table 6*** is the absence of best solutions provided by binary schemes, regardless of the usage of weights to reflect the semantics of the problem. The second aspect worth noticing is the prevalence of weighted solutions (WM and WA) that partially preserve the semantics of the original distribution, with over 69% of the provided solutions. If we wish to speedup the process of generating the optimal solution by reducing the number of considered weighting schemes, the binary weighting methods are the best candidates for exclusion from the used set of schemes.

The proposed level of partitioning of a system may differ from the original solutions to the proposed ones.

H_{e0}: *The number of modules in the proposed best solution is similar to the one in the original solution.*

H_{e1}: *The number of modules in the proposed best solution is significantly different to the one in the original solution.*

As neither of the variables has a normal distribution, we use a nonparametric procedure (Wilcoxon Signed-Rank Test) used with two related variables to test the null hypothesis that the two variables have the same distribution. This test takes into account information about the magnitude of differences within pairs and gives more weight to pairs that show large differences than to pairs that show small differences. The test statistic is based on the ranks of the absolute values of the differences between the two variables. ***Table 7*** presents the test's results. A significance analysis for this test is presented in ***Table 8***.

	N	Mean Rank	Sum Ranks
Negative Ranks	7	9.07	63.50
Positive Ranks	16	13.28	212.50
Ties	0		
Total	23		

Table 7 – Proposed Modules – Original Modules.

	Modules – Original Modules
Z (based on negative ranks)	-2.267
Asymp. Sig. (2-tailed) P	.023

Table 8 – Wilcoxon Signed Ranks Test Statistics.

Negative ranks occur when the proposed number of modules is less than in the original solution. Conversely, **positive ranks** are counted when the number of modules increases. If the number of modules is exactly the same both in the original solution and in the proposed one, we have a **tie**.
16 out of 23 proposals increase the number of modules thus favoring a more fragmented modularization scheme. The P value of 0.023 is below $\alpha = 0.05$. We can reject our null hypothesis, as there is evidence to show that the proposed modularization approach leads to a different number of modules, in general. The negative value of Z denotes that the number of modules usually increases. If we further look into the constitution of the systems where the number of modules has decreased, most of them have an AMM very close to 1, suggesting that little modularization had been achieved for these systems in the original solution. The majority of the studied systems, however, were under-partitioned, according to our criteria.

6 CONCLUSIONS

Software engineering gurus continue to sustain the ideal of minimal coupling and maximal cohesion. We have found that this desideratum does not match practitioners' reality at least in what concerns the modularity of object-oriented systems. Finding out which were the

modularization driving forces is an exercise that may prove fruitless since it is not feasible to understand the problem domain of every system in our sample. Instead, we proposed an approach that uses cluster analysis techniques and class-coupling information to obtain better modularization solutions as far as coupling and cohesion are concerned. This approach mainly considers structural coupling, but it also uses part of the semantics behind the original systems modularization.

In our quest to find the best solution we tried several clustering methods and several ways of combining the coupling information. We have found out that some of them could be discarded in future experiments.

The approach presented here can be used both in the initial design phase and in the reengineering of object-oriented legacy systems, allowing to identify ill-defined modularization situations and proposing alternative ones. The tool that we built for this purpose proved to be a valuable support to this process.

Going back to the issue of modularization criteria, other driving forces besides structural coupling and cohesion, such as semantic categorization, can be, and indeed are, often used. This may be particularly true for class libraries, as it can be easily ascertained by looking at the Java class libraries [9]. In such cases, a reorganization based on structural criteria, might lead to a confusing API.

7 RELATED AND FUTURE WORK

7.1 Related work

Clustering analysis has been successfully used to deal with the modularization problem in procedural systems [22-25]. Yet, we have not found researchers that have explored the usage of these techniques with OO technology.

On the realm of OO modularity evaluation, there are some techniques such as the one introduced in [26], where the coupling strength of MERODE [27] domain object classes is used in the evaluation. Neural networks have also been proposed as a tool to software modularization [28]. Metrics for evaluating modularity were proposed in [29]. These works in the OO field do not use cluster analysis.

Although with a different aim (finding objects in legacy code), the works presented in [30, 31] are also related to the one presented here, since they also use automatic clustering techniques.

7.2 Future work

In this experiment we have not yet considered “external” couplings, that is, those relative to components beyond the set of modules under consideration. These may correspond, among other things, to dependencies on operating system features, middleware connections, use of library components, etc. We plan to expand MOTTO capabilities in order to encompass such situations, instead of only considering self-contained systems.

The work presented here does not consider several layers of modularization (i.e. modules within modules), an important feature for very large systems. Each nested layer may use different modularization criteria. Here we could argue that our approach can be applied in a stepwise manner, by first splitting the system in a given number of subsystems (e.g. fixing its number), and then applying the same approach again within each subsystem, to identify nested modules. Given the availability of a very large system, we will build a case study.

In forthcoming experiments we also plan to try other clustering methods, namely hierarchical divisive ones. Divisive methods start with all cases in one cluster (all classes in one module, in our domain semantics). At every step, existing clusters are split until the desired number of clusters is reached. Once a cluster is split in two, the two resulting clusters cannot be merged with other clusters, but only be split again.

We will also experiment with the K-Means clustering method. This method attempts to identify relatively homogeneous groups of cases based on selected characteristics, using an algorithm that can handle large numbers of cases. It is more efficient than hierarchical clustering methods primarily because it does not compute the distances between all pairs of cases.

Last, but not the least, we plan to apply some Operations Research techniques to the multiple-decision criteria problem described in this paper.

REFERENCES

[1] ISO9126, “Information Technology - Software Product Evaluation - Software Quality Characteristics and Metrics,”. Geneva, Switzerland: International Organization for Standardization.

[2] C. Ghezzi, M. Jazayeri, and D. Mandrioli, Fundamentals of Software Engineering. Englewood Cliffs, NJ., USA: Prentice Hall, 1991.

[3] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard, Object-Oriented Software Engineering- A Use Case Driven Approach. Reading, MA, USA / Wokingham, England: Addison-Wesley / ACM Press, 1992.

[4] I. Sommerville, Software Engineering, 6th ed: Addison-Wesley Longman, 2000.

[5] R. S. Pressman, Software Engineering: A Practitioner's Approach (European Adaptation), 5th ed: McGraw-Hill Book Company, 2000.

[6] B. Meyer, Object-Oriented Software Construction, 2nd ed. Upper Saddle River, NJ, EUA: Prentice Hall PTR, 1997.

[7] B. Stroustrup, The C++ Programming Language, 3rd ed. Reading, Massachusetts, USA: Addison-Wesley

Publishing Company, 1997.

[8] M. Cantù, Mastering Delphi 2 for Windows 95/NT: Sybex, 1996.

[9] J. Gosling and F. Yellin, The Java Application Programming Interface, vol. #1 (Core Packages) / #2 (Window Toolkit and Applets). Reading, Massachussets, USA: Addison-Wesley, 1996.

[10] S. Lewis, The Art and Science of Smalltalk: Prentice-Hall, 1995.

[11] D. F. D'Souza and A. C. Wills, Objects, Components and Frameworks with UML: The Catalysis Approach. Reading, Massachussets: Addison Wesley Longman, 1998.

[12] B. Meyer, Object Success - A Manager's Guide to Object Orientation, its Impact on the Corporation, and its Use for Reengineering the Software Process: Prentice Hall International, 1995.

[13] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, Object-Oriented Modelling and Design. Englewood Cliffs, NJ, EUA: Prentice Hall, 1991.

[14] G. Booch, I. Jacobson, and J. Rumbaugh, “UML Semantics,” Rational Software Corporation Version 1.0, January 1997.

[15] G. Booch, Object Oriented Analysis and Design with Applications, 2nd ed. Redwood City, LA, USA: The Benjamin Cummings Publishing Company Inc, 1994.

[16] M. Page-Jones, What Every Programmer Should Know about Object-Oriented Design. New York, USA: Dorset House, 1995.

[17] F. B. Abreu, G. Pereira, and P. Sousa, “A Coupling-based Approach to Reengineer the Modularity of Object-Oriented Systems,” presented at 4th European Conference on Software Maintenance and Reengineering (CSMR'2000), Zurich, Switzerland, 2000.

[18] L. Kaufman and P. J. Rousseeuw, Finding Groups In Data: An Introduction To Cluster Analysis: John Wiley and Sons, 1990.

[19] F. B. Abreu and J. S. Cuche, “Collecting and Analyzing the MOOD2 Metrics,” presented at ECOOP'98 Workshop - Object-Oriented Product Metrics for Software Quality Assessment, Brussels, Belgium, 1998.

[20] F. B. Abreu, L. M. Ochoa, and M. A. Goulão, “The GOODLY Design Language for MOOD2 Metrics Collection,” presented at ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering, Lisboa, Portugal, 1999.

[21] B. Conner, D. Niguidula, and A. v. Dam, Object-Oriented Programming in Pascal - a Graphical Approach: Addison-Wesley, 1995.

[22] D. H. Hutchens and V. R. Basili, “System Structure Analysis: Clustering with Data Bindings,” IEEE Transactions on Software Engineering, vol. 11, pp. 749-757, 1985.

[23] R. W. Schwanke, “An Intelligent Tool For Re-enginnering Software Modularity,” presented at 13th International Conference on Software Engineering (ICSE'91), 1991.

[24] T. Kunz, “Evaluating Process Clusters to Support Automatic Program Understanding,” presented at 4th Workshop on Program Comprehension, 1996.

[25] G. M. Pereira, “Reengenharia da Modularidade de Sistemas de Informação,” in Departamento de Engenharia Electrotécnica e Computadores. Lisboa, Portugal: master thesis, IST / UTL, 1998.

[26] G. Poels, “Evaluating the Modularity of Model-Driven Object-Oriented Software Architectures,” presented at ECOOP Workshop on Techniques, Tools and Formalisms for Capturing and Assessing Architectural Quality in Object-Oriented Software, Brussels, Belgium, 1998.

[27] M. Snoeck and G. Dedene, “Existence Dependency: The Key to Semantic Integrity Between Structural and Behavioral Aspects of Object Types,” IEEE Transactions on Software Engineering, vol. 24, 1998.

[28] R. W. Schwanke and S. J. Hanson, “Using Neural Networks to Modularize Software,” Machine Learning, 1993.

[29] P. Predonzani, G. Succi, and A. Valerio, “Object-oriented frameworks: architecture adaptability,” presented at ECOOP Workshop on Techniques, Tools and Formalisms for Capturing and Assessing Architectural Quality in Object-Oriented Software, Brussels, Belgium, 1998.

[30] J.-F. Girard, R. Koschke, and G. Schied, “A Metric-Based Approach to Detect Abstract Data Types and State Encapsulations,” presented at 12th International Automated Software Engineering Conference (ASE'97), 1997.

[31] J.-F. Girard and R. Koschke, “A Comparison of Abstract Data Type and Object Recovery Techniques,” Science of Computer Programming, 1999.

APPENDIX I

Specification	Description
Allegro	Port of the Allegro game programming library to the SmallEiffel Eiffel Compiler.
Bast	Object-oriented framework for building fault-tolerant distributed applications.
Blox	GUI building block tool kit. It is an abstraction on top of the platform's native GUI toolkit that is common across all platforms.
BoochTower	Library of structure components such as bags, graphs, lists, stacks, strings, trees...
Cxtnsn	C based extensions to GNU Smalltalk.
Dinamico	Abstract data types library in Delphi.
Ems	Lexical analyzer that takes a list of classes in input and returns the analysis to the standard output.
Fec	Eiffel compiler for Sparc Stations.
Funcao3d	3D Function Viewer in Delphi.
Gma	Graphics package developed in Delphi.
GNU_SmallTalk	Implementation of Smalltalk-80 under GNU's public license.
Gobo	Eiffel libraries portable across various Eiffel compilers including: a Kernel Library, a Structure Library and an Utility Library.
GPCardGames	Implementation of card games.
ISE_Eiffel	Lexical analyzer that takes a list of classes in input and returns the analysis to the standard output.
J-eiffel	J-Eiffel is an Eiffel compiler with Java Virtual Machine code generation.
RallyPT	UML model on the organization of a Rally.
SIG_Container	Containers library for SIG Eiffel.
SIG_DateTime	Date and Time library for SIG Eiffel.
SIG_Eiffel	SIG Eiffel programming environment.
SIG_Libraries	Libraries for SIG Eiffel.
Stix	SmallTalk Interface to the X protocol layer that underlies all of X Windows.
Structure	A data structure library based on circular-linked-lists.
Yoocc	Compiler-compiler that uses an extended parse library which derives from the ISE Eiffel parse library.

Table 9 - Description of sample items

Specification	Type	Formalism	Number Classes	Number Couplings	Original Modules	Proposed Modules	Original RMD	Proposed RMD	Original MMF	Proposed MMF
Allegro	APP	Eiffel	33	3534	2	24	2.00	2.18	0.06	0.21
Bast	APP	Smalltalk	38	587	11	19	1.45	1.50	0.04	0.09
Blox	APP	Smalltalk	42	1264	24	16	10.86	9.90	0.01	0.28
BoochTower	LIB	Eiffel	130	2894	3	81	1.32	1.25	0.01	0.09
Cxtnsn	LIB	Smalltalk	20	26	5	9	3.50	3.60	0.09	0.24
Dinamico	LIB	Delphi	12	52	11	2	0.92	0.67	0.00	0.10
Ems	APP	Eiffel	111	3370	8	82	5.55	5.91	0.01	0.12
Fec	ENV	Eiffel	171	17631	6	126	2.28	2.21	0.00	0.03
Funcao3d	APP	Delphi	18	146	5	7	3.61	3.11	0.10	0.25
Gma	LIB	Delphi	46	139	18	15	4.30	4.57	0.14	0.19
GNU_SmallTalk	ENV	Smalltalk	246	6934	104	55	12.68	13.86	0.00	0.14
Gobo	LIB	Smalltalk	119	4563	5	83	2.06	2.09	0.03	0.09
GPCardGames	APP	Smalltalk	20	362	3	2	0.60	0.60	0.03	0.08
ISE_Eiffel	ENV	Eiffel	150	13210	3	128	1.66	1.71	0.01	0.03
J-eiffel	ENV	Eiffel	21	565	7	5	1.00	0.95	0.06	0.17
RallyPT	APP	UML	10	23	1	5	.000	2.50	0.10	0.31
SIG_Container	LIB	Eiffel	47	1215	2	22	1.74	1.87	0.04	0.08
SIG_DateTime	LIB	Eiffel	20	929	4	8	2.00	2.00	0.12	0.21
SIG_Eiffel	ENV	Eiffel	79	7531	4	58	2.18	2.20	0.02	0.09
SIG_Libraries	LIB	Eiffel	34	2509	3	27	2.74	2.38	0.08	0.15
Stix	LIB	Smalltalk	110	4119	52	26	15.60	17.02	0.00	0.19
Structure	LIB	Smalltalk	14	713	4	5	1.71	1.79	0.03	0.28
Yoocc	ENV	Eiffel	58	457	7	24	2.66	2.90	0.02	0.13

Table 10 - Sample metrics