Metrics for Controlling Effort During Adaptive Maintenance of Object Oriented Systems

F. Fioravanti, P. Nesi, F. Stortoni
Department of Systems and Informatics, University of Florence
Via di S. Marta 3, 50139, Florence, Italy
nesi@ingfi1.ing.unifi.it, tel: +39-055-4796523, fax: +39-055-4796363

Abstract

The object-oriented modeling has been largely adopted in industry in the last years. Several systems built 4 or 5 years ago may need an adaptive maintenance process in order to better satisfy market and customer needs. In this paper[1], a model for effort estimation/prediction of the adaptive maintenance is presented. A selection of metrics for effort estimation has been applied to the general model for evaluating maintenance effort. The metrics presented have been validated against real data. The validation presented has shown that some metrics that can be profitably employed for effort estimation/prediction can be also used with success for the estimation/prediction of the maintenance effort. Moreover, the results obtained gives some guidelines to maintain under control relevant factors for the adaptive maintenance.

Index terms: *object-oriented metrics, maintenance, effort estimation.*

1 Introduction

Traditional metrics for procedural languages can be difficulty applied for evaluating object-oriented systems [15], [23], [24]. Several interesting works on predicting and evaluating maintainability, reusability, reliability, and effort for system development and/or maintenance have been presented. This information has been in many cases related to complexity, size and sometimes to other parameters which should describe the system conformity with the object-oriented paradigm concepts. Certain relationships have been demonstrated correct by using validation processes – e.g., [24], [1], [17], [19].

In assessing object-oriented systems, it is very important to take into account the relationships of *is-part-of*, *is-referred-by*, *is-a*. These aspects must be captured with specific metrics, otherwise their related costs are not directly measurable (e.g., the costs of specialization, the costs of object reuse, the cost of development, the costs for reengineering, maintenance, etc.). The *is-part-of* relationships is very important for considering the composition/decomposition of the system. The relationships of *is-referred-by* frequently hide dynamic links that can lead to manage lists and reticular structures. The *is-a* relationship is a strong vehicle of reuse and polymorphism, but it has to be used with care since the inheritance can produce degenerative conditions in which the presence of specialization can also decrease the system reusability and maintainability. The object-oriented modeling has been largely adopted in industry in the last years. Unfortunately several systems built 4 or 5 years ago may need an adaptive maintenance process in order to better satisfy market and customer needs.

In order to guarantee the control of the development process, as well as the processes of maintenance or reengineering, quantitative metrics for evaluating and predicting system characteristics must be used. One of the most important issues that should be maintained under control is the effort (i.e., man-months or -days needed for system development or maintenance including analysis, design, test or in some cases only for coding). To this end, a linear/non-linear relationship between software complexity/size and effort is commonly assumed [22]. Therefore, the problem of effort evaluation is typically shifted to the problem of complexity or size evaluation. It should be noted that, when software complexity evaluation is performed after system building, it can be useful for: (i) predicting maintenance costs, (ii) comparing productivity and costs among different projects, (iii) learning the development process efficiency and parameters, (iv) predicting reengineering costs. When software complexity evaluation is performed before system building, it can be used for predicting costs for development, for testing, for the early maintenance, etc.

Moreover, on the basis of the knowledge, which is present in the early stages of the software life-cycle (e.g.,

[1]This work was partially supported by MURST Ex60% govern Ministry of University and Scientific Research.

number of classes, main class relationships, number of methods, method interfaces, etc.), the process of system analysis allows the definition and tuning of metrics for predicting the effort. From the cognitive point of view, the observable complexity can be regarded as the effort to understand subsystem/class behavior and functionalities. This complexity can be usually evaluated in the early phases and can be used for predicting costs of reuse and maintenance [7] or for estimating and predicting other features – e.g., [24], [27], [13], [9], [19], [14], [23], [18], [20].

In this paper, a study about the estimation and prediction of the effort for the adaptive maintenance of object-oriented systems coded in C++ is presented. To this end, a model for the effort estimation/prediction is proposed. For this model some new metrics and the adoption of well-known metrics are shown. The metrics presented have been validated against real data. The validations presented have shown that some metrics that can be profitably employed for effort estimation/prediction can be used with success also for the estimation/prediction of the maintenance effort. The metrics presented in this paper have been added to a framework specifically defined for C++ language [23], [24], [11].

This paper is organized as follows. In the Section 2, the metrics proposed for evaluating and predicting class effort due to the adaptive maintenance on the basis of complexity/size are reported. Then, in Section 3 the validation of the most important metrics proposed is presented together with a comparison against metrics extracted from the literature. Conclusions are drawn in Section 4.

2 Metrics for Estimation/Prediction of Adaptive maintenance Effort

In this Section, a new model and metric for the estimation and prediction of adaptive maintenance effort is presented. The model/metric is based on classical metrics for the estimation of development effort. To this end, a selection of metrics from the literature has been performed on the basis of the authors experience (e.g., [23], [24], [11], [10], [6], [22]) and considering the several similar experiences presented in the literature – e.g. [20], [1], [9], [4], [15], [5], [19], [26], [8] and [2].

A validation of the proposed metrics for prediction/estimation of the adaptive maintenance effort is reported in Section 3, in which they have been applied on the real case of MOODS ESPRIT IV Project. The validation process has been performed by using a multilinear regression analysis [25].

Before to present the new model/metric for estimation/prediction of the adaptive maintenance effort, some already known metrics are presented since the new model is based on the latter.

To help the reader to understand the metric formulation and discussion, the authors have prepared Tab.7, at the end of the paper, in which the metrics and their corresponding meaning are reported in alphabetic order.

2.1 Class Complexity and Size

At the method level, traditional functional metrics such as the McCabe Ciclomatic Complexity, Mc, [21], [16], the Halstead measure, Ha [12], and the number of lines of code, LOC, can be used. On the other hand, these metrics are not very suitable for evaluating object-oriented projects, since they are not capable of considering all the object-oriented aspects [27], [13], [24]. In fact, they neglect information about class specialization (*is_a*, that means code and structure reuse), and class association and aggregation (*is_part_of* and *is_referred_by*, that mean class/system structure definition and dynamic managing of object sets, respectively). On the other hand, WMC (Weighted Methods for Class) [9], and LOC (used in [20]) have been adopted as good compromises between precision and simplicity of evaluation for measuring the development effort.

In [10], [24], the fully object-oriented metric, CC, for evaluating class complexity/size has been presented together with a comparison with the above mentioned object-oriented metrics. This metric includes cognitive, structural and functional aspects. The class complexity/size, CC, has been defined as the sum of the External Class Description (ECD) (complexity/size due to class definition and method interface definition) and the Internal Class Implementation (ICI) (method implementation):

$$CC = ECD + ICI, \tag{1}$$

where CC components can be decomposed in complexities due to locally and inherited class members:

$$ECD = ECDL + ECDI, \tag{2}$$

$$ICI = w_{CL}CL + w_{CI}CI, \tag{3}$$

where:

$$\begin{aligned} ECDL &= w_{CACL}CACL + w_{CMICL}CMICL, \\ ECDI &= w_{CACI}CACI + w_{CMICI}CMICI, \end{aligned} \tag{4}$$

and thus: $CACL$ is the Class Attribute Complexity Local (complexity due to attributes locally defined); $CACI$ is the Class Attribute Complexity Inherited; $CMICL$, the Class Method Interface Complexity Local (complexity of local method parameters); $CMICI$, the Class Method Interface Complexity Inherited (as the previous for methods inherited); CL, the Class complexity/size due to Local methods;

and CI, the Class complexity/size due to Inherited methods. Metrics CL and CI measure the complexity/size of the functional part estimated by using one of the above mentioned metrics: Ha, Mc, LOC. Thus, according to the metric used, CC may be considered a complexity or a size metric.

Metric CC takes into account both structural (attributes, relationships of *is_part_of* and *is_referred_by*) and cognitive (methods, method "cohesion" by means of $CMICL$ and $CMICI$, respectively), and functional by means of CL, CI, aspects of the class. The class attributes can be (i) class instances (evaluated by considering metric CC of their corresponding class), or (ii) basic types (e.g., `char`, `int`, `float`, etc.) for which the complexity is posed to predefined values.

Weights in the above metrics have to be evaluated by using a multilinear regression [25] on the basis of the actual class effort [24]. The weight values obviously depend on the purpose for which the metric is used and on the phase in which the metric is evaluated. Thus, a trend for the weights along the development and/or maintenance and/or the reuse process has to be determined. In general, w_{CACI} and w_{CI} are typically negative stating that the inheritance of attributes and methods leads to save complexity/size and, thus, effort.

Please note that WMC [9] is the sum of all McCabe complexities of class-methods which is quite equivalent to the above CL metric evaluated by using Mc as basic metric [24]. The metric proposed by Thomas and Jacobson [27] and its evolution proposed by Henderson-Sellers [13] can be defined in terms of CC components: $TJCC = w_{CACL}CACL + w_{CL}CL$ and $HSCC = w_{CACL}CACL + w_{CL}CL + w_{CI}CI$, respectively. For these reasons, CC can be considered as a generalization of these metrics.

The above mentioned ECD metric gives a measure of what can be observed by reading the class definition, for example in the phase of class reuse or maintenance.

Metric CC can also be used for predicting class complexity/size. In particular, the prediction is performed by considering only the class definition: attributes declarations and methods prototypes. This estimation can be performed during system analysis/early-design. Therefore, the predictive version of CC metric has the following form:

$$CC' = w_{CACL'}CACL' + w_{CMICL'}CMICL' + w_{CACI'}CACI' + w_{CMICI'}CMICI', \quad (5)$$

where: $CACI'$ and $CACL'$ are obviously estimated on the basis of the CC' of class members. Even in this case, the weights must be evaluated by using a validation process such as that reported in the next section.

2.2 Class Attributes and Methods

A lighter approach for class size evaluation can be simply based on counting the number of local attributes and methods (see metric $Size2 = NAL + NML$ defined by Li and Henry in [19], sum of the number of local attributes and methods). On the other hand, the counting of class members could be in many cases a too coarse measure. For example, when a class attribute is an instance of a very complex class, often implies a high cost of method development, which is not considered simply by increasing NAL of one unit. Moreover, $Size2$ does not consider the class members inherited (that is, reuse). In order to improve the metric precision, a more general metric has been defined by considering the sum of the number of class attributes and methods both locally defined and inherited, respectively:

$$NAM = NAML + NAMI, \quad (6)$$

therefore, NAM can be expanded assuming the form:

$$NAM = w_{NAL}NAL + w_{NML}NML + w_{NAI}NAI + w_{NMI}NMI. \quad (7)$$

Also in this case, the typical values of weights can be estimated by using a multilinear regression technique. If the purpose is to detect critical conditions (of excessive cost of reuse or development) the weights can be imposed to be equal to 1 [24]. Please note that, NAM metric can be used since the early phases of software development for predicting class size and thus class development costs.

2.3 Metrics for Adaptive Maintenance Effort

As it has been demonstrated in [24], the above mentioned metrics for complexity/size are strongly correlated with the development effort. In this section, a new model/metrics for the estimation and prediction of the class effort for the adaptive maintenance is proposed.

The class effort for the adaptive maintenance of a system is typically spent to perform several operations: the comprehension of the system, the addition of functionalities, the adaptation of selected parts. At the level of code, the *adaptation of selected parts* can be decomposed in deleting some parts and adding other portions. Typically, if the team who performs the adaptive maintenance is the same that has generated the application under adaptation, the effort for comprehending the system can be neglected with respect to other factors. Thus, in these conditions, the effort is mainly due to the addition and deletion of code pieces.

Therefore, considering: (i) a metric M for class size, complexity or volume estimation, typically related to effort;

(ii) M_a, the value obtained evaluating metric M after the adaptive maintenance; (iii) M_b, the value obtained evaluating metric M before the adaptive maintenance; then, the complexity/size newly enforced in a class during the adaptive maintenance, M_{am}, can be expressed as:

$$M_{am} = M_a - M_b - m_d,$$

where: M can be one of the above discussed metrics such as $CC, CC', WMC, TJCC, HSCC, NAM, Size2$; and, m_d takes into account the class code that has been removed during the adaptive maintenance. In some cases, m_d may be very difficulty estimated directly on the code. If LOC is used as M, the estimation of the number of removed lines of code can be an approximated solution to estimate m_d. Unfortunately, this approach cannot be used when more complete and complex metrics such as CC are used as M, since structural and functional aspects and the class definitions and relationships should be considered. On the other hand, m_d in the adaptive maintenance should be a small percentage of the class code, thus, it can be approximated with wM_b, obtaining the following metric:

$$M_{am} = M_a - w_{M_b} M_b, \tag{8}$$

where

$$w_{M_b} = 1 + w.$$

Typically, the programmers are quite reluctant towards the deletion of methods since in many cases it may be difficult and time consuming to be sure that they are not used by other classes or by other team members. Thus, the code deletion in the classes is typically limited to parts of methods, and only in some cases to entire methods and attributes. This produces a well-known maintenance problem for object oriented systems: *the presence of non-used methods and attributes*.

If the team that has developed the original system is different with respect to that performing the adaptive maintenance a term for considering the effort of comprehension has to be added in equation (8).

2.3.1 Class Complexity and Adaptive Maintenance Phases

Considering $M \equiv CC$ in equation (8) the complexity/size enforced in the system during the adaptive maintenance, CC_{am}, can be expressed as:

$$CC_{am} = CC_a - w_{CC_b} CC_b.$$

Therefore, since CC is defined as the weighted sum of six terms, the whole metric CC_{am} can be regarded as a metric with 12 terms and their corresponding 12 weights, 6 for CC_a and 6 for CC_b. w_{CC_b} is practically included into to the weights of CC_b that change their meaning. Moreover, since CC_b also takes into account some cognitive aspects of the system, CC_{am}, can also partially model the effort for comprehending the system before adaptation.

Therefore, a complete validation of CC_{am} metrics has to be performed by considering the whole structure of CC. In that case, a multilinear regression can be used for estimating suitable weights. This validation also allows to identify terms of metric CC_{am} that are relevant for effort estimation of the adaptive maintenance.

The above approach can be used for defining metrics for the adaptive maintenance on the basis of the above mentioned classical metrics for effort estimation: $TJCC_{am}$, $HSCC_{am}$, $Size2_{am}$, etc. In the next section, the adoption of these metrics in equation (8) for effort estimation is compared with the results obtained for CC_{am}. These metrics can be useful for: (i) comparing productivity and costs among different projects, (ii) learning the development process efficiency and parameters.

A predictive version of CC_{am} can be obtained by considering CC' into equation (8):

$$CC'_{am} = CC'_a - w_{CC'_b} CC'_b.$$

Also in this case, a multilinear regression can be used for estimating the suitable 8 weights. Once the system analysis of the adaptation phase is performed, the structures of classes at the end of the adaptive maintenance are known. With this knowledge, it is possible to use CC'_{am} metric for predicting the costs for the adaptive maintenance.

2.3.2 Class Members and Adaptive Maintenance Phases

Considering $M \equiv NAM$ in equation (8) the size enforced in the system during the adaptive maintenance, NAM_{am}, can be expressed as:

$$NAM_{am} = NAM_a - w_{NAM_b} NAM_b.$$

Even in this case, weight w_{NAM_b} has to be estimated by a validation process (see Section 3). A complete validation has to be performed by considering the whole structure of NAM for both NAM_a and NAM_b for estimating the 8 weights.

Therefore, once a draft analysis of the adaptation phase is performed, the number of members that the classes will present at the end of the adaptive maintenance is known. Exploiting this early knowledge, it could be possible to use NAM_{am} metric for predicting the effort for the adaptive maintenance.

3 Metrics Validation

A metric analysis has been performed in order to identify which metrics among the above-mentioned are better ranked for evaluating and/or predicting the effort for adaptive maintenance of a system.

The comparative analysis with validation has been carried out among the previously defined metrics and some of those already defined in the literature. Moreover, since most of the new metrics defined are very complex and computationally expensive to be evaluated, an analysis to verify the influence of their parameters in producing the final result has been performed in order to identify the minimum number of parameters which are needed to obtain suitable measures. Therefore, the analysis performed is more than a simple validation since it has produced a clear view of the above mentioned metrics for estimating class effort of the adaptive maintenance.

The validation has been performed by considering the data relative to MOODS ESPRIT IV project (Music Object Oriented Distributed System) (*http://www.dsi.unifi.it/~moods*) concluded in November 1998. The project consisted in the implementation of a distributed systems of music lecterns starting from a stand-alone music editor (LIOO, Lectern Interactive Object Oriented) according to the technology transfer approach of HPCN project managed via Technology Transfer Nodes, TTNs. The project addressed the adaptation of a stand-alone music lectern and editor to transform it in a distributed system of music editors, for cooperative management of music. The reference TTN was TETRApc having as partners CPR (Consortium Pisa Research), CESVIT (High Tech Agency).

MOODS has been a multipartners project evaluated about 44 man months including hardware and software. Hardware effort was mainly devoted for implementing special music lecterns. Software effort was used for (i) porting the stand alone music editor from MS-DOS to UNIX, (ii) adapting the original version of music editor towards a distributed co-operative version, (iii) implementing an independent orchestra network configurator and monitor (called ONCM), (iv) implementing several additional advanced functionalities for the cooperative music editor, (v) connecting the music editors to a database of music scores (called MOODSDB).

The data and the code used in the following validations are referred to points (i) and (ii), which are the adaptive maintenance phases. In MOODS project, the firsts workpackages have been the porting and the adaptation to distributed approach, (i) and (ii); then phases (iii), (iv) and (v) have been performed in parallel. Phases (i) and (ii) have been performed by the same team that developed the original version of LIOO. This has strongly reduced the effort for class and documentation comprehension. LIOO was comprised of 113 classes and these have been become 133 after performing phases (i) and (ii), with 9.6 man months. This version has been called LIOO5 since several intermediate versions were analyzed.

	Before (LIOO1)	*After* (LIOO5)
NCL	113	133
NRC	19	25
TNM	1087	1341
TLOC	12150	13891
MCC	876	877
MNA	7	7
MNM	38	39

Table 1. Overview of the system before and after the adaptive maintenance process.

MOODS project has been built by using object-oriented analysis and design methodologies (i.e., Booch [3]), and all the project phases have been monitored and maintained under control by using all the metrics presented above and several others (e.g., those defined and discussed in [24], [11], [10] and [6]).

Before presenting the validation of the proposed metrics, the analysis of the system evolution from its early version to the project completion including the phases of porting and adaptive maintenance is presented.

3.1 Analysis of System Evolution

In order to better understand the system evolution, in Tab. 1 a summary of some system level metrics is reported for both LIOO1 *(before)* and LIOO5 *(after)* versions. The process of adaptive maintenance has provoked an increment of about 15% in the system size/complexity, because of the addition of new funcionalities. This increment resulted well distributed on system classes since the mean values of CC, NA and NM have not changed their values in a significant way after the adaptation. By the analysis of NRC results that 6 roots or stand-alone classes have been added. The added classes in our example are stand alone classes for representing the main score (e.g., class *Partitura*). In LIOO1, some classes having a CC very high were identified (e.g., CC about 4300). This condition was corrected during adaptive maintenance in version LIOO5 in which the maximum CC is about 3200.

In Fig.1, the trend of metric CC' for complexity prediction/estimation (in log scale) is reported together with the evolution of the actual effort. The trend analysis shows also the predicted values obtained by using extrapolation algorithms, e.g., for predicting the cost of designing and coding in the phase of analysis, etc. The figure considers the complexity evolution from the detailed analysis to the 95% of

the project completion.

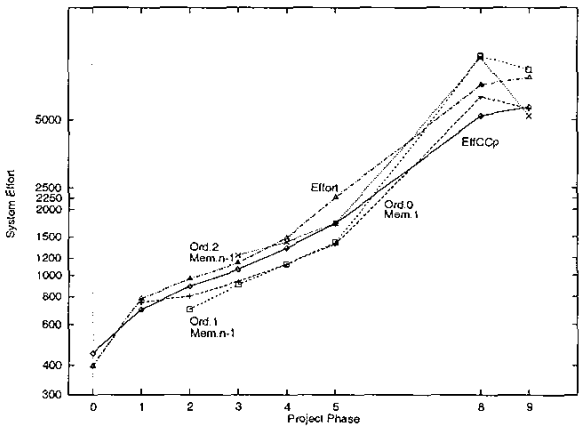

Figure 1. Trend of the actual effort and the predicted effort of the whole system on the basis of CC' class effort prediction. Several different algorithms for predicting values on the basis of the values estimated in the previous phases are reported.

Another interesting trend analysis is that of the metric weights. In Fig.2, the trend for weights of CC of the project is reported (the weights have been estimated during the validation of CC against the real effort of development). This graph is referred to the same project and in the same temporal windows as those discussed for Fig.1. The project presents from phase 1 to 5, the period in which has been performed the porting and adaptive maintenance. From Fig.2, it can be observed that:

- w_{CMICL} tends to increase quite linearly with the project phase. The corresponding metric is becoming lower with respect to other factors;
- w_{CL}, w_{CACL} are quite constant. The local complexity due to functional and structural aspects of the classes is always relevant in the same manner along the process even during the porting and adaptation;
- w_{CI} is negative and decreasing (this is much more evident after the adaptation phase). The corresponding metric term is a gain for the whole complexity. This means that the exploitation of inheritance is observed as a decrement especially in development costs.
- w_{CMICI} presents a negative trend from phase 1 to 5. During the porting and adaptation, the corresponding metric has been more relevant and almost comparable with respect to metrics related to local factors. This means that during the porting and adaptation the presence of detailed interfaces for inherited methods was a source of save.
- w_{CACI} presents a positive trend from phase 1 to 5. During the porting and adaptation the corresponding metric has been lower with respect to metrics related to local factors. This is due to the fact that the porting and adaptation has been implemented by restructuring functional parts, even for those involved in the inherited part, thus the inspection of the attributes of the super-classes has been a cost.

This example has shown how the weights trend analysis can give interesting suggestions about the system evolution. Please note that the above graph reporting the evolution of LIOO in MOODS project, it includes the configurator, ONCM, and not the database connection, MOODSDB.

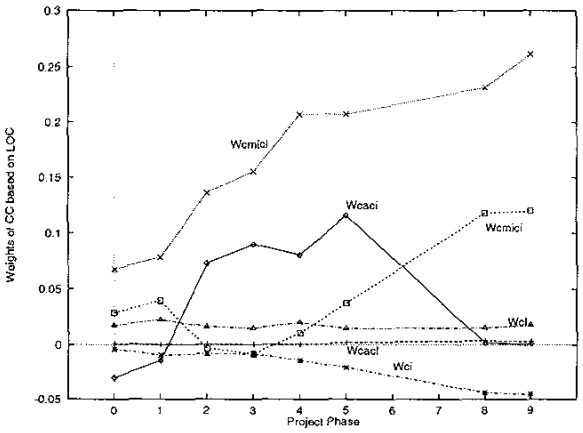

Figure 2. Trend of the weights of metric CC.

3.2 Validation process

According to the model defined may exist some exceptions that can lead to obtain non correct estimations; since in a system under adaptive maintenance:

1. some classes in the early version can be fused into one in the new version. This is a quite rare operation during an adaptive maintenance.
2. the functionalities managed by a class in the early version can be distributed in more classes in the new version. This is quite frequent operation when a class presents high dimensions.
3. some completely new classes can be added for addressing the new functionalities. These have to be considered totally new classes, $M_b = 0$.

The distribution of the changes along system classes is typically not uniform. On the other hand, the grouping of

classes with similar needs of adaptation is a solution for getting more precise results. In the validation reported in the following no grouping has been performed, since the system evolution has been only marginally affected by the above reported problems. For example, in LIOO, there were several classes representing musical symbols that according to the adaptation analysis have been marginally modified and all in a similar manner. In the same system, there were classes for managing the user interface that has been quite uniformly changed for passing from the local to the distributed version of the system.

Moreover, it is very difficult to collect data by considering projects that have been evolved in a similar way. Thus, the validation has been performed by using only MOODS project. The validation process has been performed by using a multilinear least median squares technique [25] considering (i) the relationship between effort and metrics as linear, (ii) the values of direct metrics, and (iii) the weights as unknown. Considering that the effort for each class of the system was available, the metric validation has been based with more than 120 degrees of freedom. This confers to the validation process a certain relevance.

The complexity/size metrics can be classified in *a-posteriori* and predictive metrics. The main difference between these two classes of metrics is the availability or not of the C++ code. In general, predictive metrics are evaluated only on the basis of the class interface (e.g., C++ Header files), while *a-posteriori* metrics need also class implementations (method implementation).

In general, *a-posteriori* metrics consider all the class aspects: attributes, methods interface and method implementation (both locally defined and inherited). Predictive metrics can be evaluated also if the implementation phase has not been performed yet, such as in the early phase of system development. Thus, these metrics can also be used to predict the adaptive maintenance effort by knowing only the interface that classes will have at the end of the system adaptation. This can be very useful for evaluating the adaptive maintenance effort needed during the planning of adaptation.

The validation/analysis of metrics for these two categories are separately discussed in the next subsections.

By using real effort adaptive maintenance data (provided in man/hours), the weights for CC_{am}, CC'_{am}, and NAM_{am} have been calculated. These weights have been calculated by minimizing the *least median of squares* like robust estimator [25]. The estimator represents a good compromise between robustness with respect to outliers and the reaching of correlation optimization.

The correlation values have been reported with all the data evaluated in order to have an immediate evaluation of the goodness of the results obtained.

CC_{am}			
	w	$\mid t-value \mid$	$p-value$
CL_b	-0.012	2.32	0.022
CI_b	-0.024	1.15	0.250
$CMICL_b$	0.522	9.02	0.000
$CMICI_b$	0.009	0.18	0.860
$CACL_b$	-0.009	1.95	0.053
$CACI_b$	0.124	1.11	0.267
CL_a	-0.022	3.38	0.001
CI_a	-0.032	1.94	0.055
$CMICL_a$	0.545	11.12	0.000
$CMICI_a$	0.037	0.94	0.348
$CACL_a$	-0.008	1.79	0.076
$CACI_a$	0.211	2.13	0.035
Correlation	0.84 with all components 0.87 by removing $CMICI$		
Variance	116		
R-squared	0.825		
F-stat (*p-value*)	47.11 (0.000)		

Table 2. Results of the multilinear regression analysis for *effort evaluation* of classes by using CC_{am} metric. Values of weights and their corresponding confidence values are reported.

3.3 A-posteriori estimation

In Tab.2, the results of the multilinear regression analysis is reported. It has been carried out by considering the real adaptive maintenance effort in man-hours and the 12 CC_{am} metric components, by using the techniques discussed in [25].

In Tab.2, some statistical values that shows the confidence of the weights are reported for each component of CC_{am}. In order to understand the effectiveness of the results, the *Student test* has been performed. Relevant coefficients have to provide a *t-value* greater than $t_{n-p,1-\alpha/2}$, where n is the number of classes, p is the number of coefficients to be estimated and α is the percentage of confidence (i.e., for α equal to 0.05 a confidence interval of 95% has been obtained). In order to avoid the recovering of data from probability tables also *p-values* are reported. *p-value* represents the probability that a *Student-t* with $n-p$ degrees of freedom becomes larger in absolute value than the *t-value* obtained. Below the rows containing the weights, the correlation between the real adaptive maintenance effort and the estimated CC_{am} with the identified weights, is reported. In order to evaluate the degree of confidence of the regression process *R-squared* and the relative *F-Test* (F-stat) results are reported.

In order to understand the results, it must be noted that a *Student-t* can be considered quite similar to a Gaussian curve if the number of degrees of freedom is greater than 30. In our assessment $n - p > 100$, and therefore the absolute value of *t-value* has to be greater than 1.96 for obtaining a confidence interval of 95%. On the other hand, in

CC'_{am}	w	$\mid t-value \mid$	$p-value$
$CMICL'_b$	0.476	8.70	0.000
$CMICI'_b$	0.122	1.84	0.068
$CACL'_b$	-0.038	4.16	0.000
$CACI'_b$	-0.124	0.76	0.449
$CMICL'_a$	0.492	11.83	0.000
$CMICI'_a$	0.102	1.77	0.079
$CACL'_a$	-0.033	3.94	0.000
$CACI'_a$	-0.044	0,27	0.791
Correlation	0.79 with all coefficient 0.81 by eliminating $CACI'$ component		
Variance	126		
R-squared	0.708		
F-stat (*p-value*)	44.35 (0.000)		

Table 3. Results of the multilinear regression analysis for *effort prediction* of classes by using metrics CC'_{am}: values of weights and their corresponding confidence values are also reported.

NAM_{am}	w	$\mid t-value \mid$	$p-value$
NAL_b	1.162	0.77	0.440
NAI_b	2.194	1.34	0.182
NML_b	1.399	3.73	0.000
NMI_b	-0.587	1.83	0.069
NAL_a	3.485	2.46	0.015
NAI_a	1.280	0.85	0.396
NML_a	1.501	4.62	0.000
NMI_a	-0.264	0.88	0.380
Correlation	0.75		
Variance	191		
R-squared	0.729		
F-stat (*p-value*)	41.813 (0.000)		

Table 4. Results of the multilinear regression analysis for *effort prediction* of classes by using metric NAM_{am}: values of weights and their corresponding confidence values are also reported.

our opinion, a term like $CACL_a$ having a probability lower than 8% that $t_{n-p,1-\alpha/2}$ is greater than 1.79 could be acceptable, because it means that a confidence interval greater than 92% has been obtained. Another important consideration arises from the fact that we cannot eliminate only a term *before* or *after* adaptive maintenance, but we must proceed to eliminate or consider valid both the terms (*after* and *before*), otherwise the CC_{am} metric could lose its structure.

According to the above considerations, only $CMICI$ can be removed from CC_{am} structure because both $CMICI_b$ and $CMICI_a$ satisfy the null hypothesis. By imposing a null value (0) to these coefficients the correlation increases to 0.87, while eliminating any other couple of terms the correlation decreases.

Since the model presented for the estimation of the adaptive maintenance takes into account the weighted difference between similar terms (e.g., $w_{CMICL_a}CMICL_a - w_{CMICL_b}CMICL_b$), the coefficient sign must be carefully considered. In effect, only $CMICL$ and $CACI$ terms increase the effort, while the other low-level metrics give a negative contribution. This result is in conformance with the typical object oriented paradigm guidelines; in fact, the consequences of the addition of an attribute near to a root class can provoke a great effort for adapting the subclasses that have to manage a bigger quantity of information. On the other hand, the effort for managing this information is frequently demanded to method interfaces of leaf classes adding complexity to the $CMICL$ term.

3.4 Predictive estimation

In Tab.3, the results obtained for metric CC'_{am} are reported. Also in this case both the weights for all the components and the overall statistic of the metric are reported.

It should be noted that the correlation reported for CC'_{am} is very close to those reported for CC. Analyzing the statistical values related to all the components, it is evident that $CACI'$ satisfies the null hypothesis and, therefore, the relative coefficient should be *zeroed*. A new evaluation of the correlation after removing $CACI'$ term raises the correlation to 0.81 with a variance of 118. Even also if the more complete metric, CC_{am}, is better ranked with respect to the reduced CC'_{am}. This can give a very good prediction of the effort for the adaptive maintenance.

$CMICL'$ components is the most significant term of the regression process, focusing on the fact that in the reengineering phase a great attention should be devoted to the method interface.

Metric NAM_{am}, can be suitably and frequently used for effort prediction [24]. In this particular case, it has been used for predicting the effort for the adaptive maintenance. In Tab. 4, the results regarding the correlation with effort are reported. A relatively strong value of correlation has been obtained for this metric. Please note that this metric is very cheap to be calculated, and the high correlation obtained means that it can be a very good first approximation of adaptive maintenance effort, even if the methods interface of the adapted system has not been defined. The analysis performed on NAM_{am} evidences that NML is the most important factor among the metric components. This is a sort of confirmation of the considerations about $CMICL$ component of CC_{am}. Thus, NAM_{am} can be profitably used in the early first phase of reengineering analysis; once the methods interface is defined CC'_{am} can be used to have a better approximation of the effort. As soon as the adaptive maintenance phase has been completed CC_{am} and CC values can be adopted to evaluate the correct coefficients to be applied in future estimations, or to evaluate the productivity of working group that has worked on the adaptive mainte-

	A-Posteriori Metrics		
	CC_{am}	$TJCC_{am}$	$HSCC_{am}$
Max Correlation	0.87	0.42	0.43
Variance	166	494	2043
Number of weights	10	2	6
	Predictive Metrics		
	CC'_{am}	NAM_{am}	$Size2_{am}$
Max Correlation	0.81	0.75	0.73
Variance	118	191	59
Number of weights	6	8	1

Table 5. Comparison of the metrics identified for the adaptive maintenance effort.

	CC_a	CC'_a	NAM_a	$Size2_a$
Correlation	0.64	0.56	0.67	0.52
Variance	90	87	166	210
Number of weights	6	4	4	0

Table 6. Correlation between pure effort estimation metric and maintenance effort.

nance. This method can be very encouraging because the cost of obtaining values for these metrics since the early phases of software life-cycle is very low. Therefore, small errors may be accepted in the predictive estimation of effort by using NAM metric.

3.5 Comparison between metrics in the literature

In Tab.5, a comparison among the metrics defined in the literature and those proposed in [24] for effort estimation is reported. The comparison has been performed by using all these metrics as metric M in equation (8) for the estimation of the adaptive maintenance effort, obtaining in this way a set of new metrics: CC_{am}, $TJCC_{am}$, $HSCC_{am}$, CC'_{am}, NAM_{am}, $Size2_{am}$.

The results have shown that metrics $TJCC_{am}$ and $HSCC_{am}$ are not very suitable; because they do not take into account the methods interface that have great influence in the calculation of adaptive maintenance effort. On the other hand, metric $Size2_{am}$ seems to have a relative high correlation, considering that in $Size2_{am}$ only the local class members are considered. This fact is strengthen by the consideration that in NAM_{am} also the more significant terms address the local part of the class.

Tab.6 presents the values obtained for traditional development effort metrics when these are used for the estimation of the effort for the adaptive maintenance. These values have been obtained only via validation by considering the code of LIOO5 (after the adaptive maintenance) and the real effort for the maintenance. The correlations obtained between maintenance effort with CC_a, NAM_a and $Size2_a$ are not satisfactory. The results demonstrate that classical metrics for estimating and prediction development effort are unsuitable for the estimation of the adaptive maintenance effort with respect to the newly identified metrics.

metric	comment
$CACI$	Class Attribute Complexity/size Inherited
$CACL$	Class Attribute Complexity/size Local
CC	Class Complexity/size
CI	Class Method complexity/size Inherited
CL	Class Method complexity/size Local
$CMICI$	Class Method Interface Complexity/size Inherited
$CMICL$	Class Method Interface Complexity/size Local
ECD	External Class Description
$ECDI$	External Class Description Inherited
$ECDL$	External Class Description Local
Ha	Halstead metric [12]
$HSCC$	Class Complexity by [13]
ICI	Internal Class Implementation
LOC	number of Lines Of Code
M_{am}	Generic Metric for Maintenance
Mc	McCabe ciclomatic Complexity
MCC	Mean value of Class Complexity
MNA	Mean Number of Class Attributes
MNM	Mean Number of Class Methods
NOC	Number Of Children [9]
$NSUB$	Number of SUBclasses
$NSUP$	Number of SUPerclasses
NA	Number of Attributes of a class
NAI	Number of Attributes Inherited of a class
NAL	Number of Attributes Locally defined of a class
NM	Number of Methods of a class
NMI	Number of Methods Inherited of a class
NML	Number of Methods Local of a class
NAM	Number of Attributes and Methods of a class
$NAMI$	Number of Attributes and Methods Inherited of a class
$NAML$	Number of Attributes and Methods Locally defined of a class
NCL	Number of CLasses in the system
NRC	Number of Root Classes in the system class tree
$Size2$	Number of class attributes and methods [19]
$TJCC$	Class Complexity by [27]
$TLOC$	Total (System) Number of LOC
TNM	Total (System) Number of Methods
WMC	Weighted Methods for Class [9]

Table 7. Glossary of the metrics mentioned in this paper.

4 Conclusions

In this paper, a general model for adaptive maintenance effort evaluation and prediction has been proposed. The application of the model to metrics for total effort estimation and their validation with respect to real data collected during the ESPRIT project MOODS has allowed to identify a suitable set of new metrics for maintenance effort estimation and prediction. The metrics analyzed and validated have been compared with well known complexity metrics for the estimation of development effort presented in the literature. Statistical validations and the estimated weights, evaluated with a robust estimator, have been reported. Main lessons learned from this analysis can be summarized in the following points: (i) during adaptive maintenance a strong

attention should be paid to methods definition and in particular to the methods interface; (ii) attributes added in the higher part of the hierarchy should be carefully considered in the analysis phase, (iii) CC'_{am} resulted to be the most suitable metrics for predicting the effort for the adaptive maintenance, (iv) the complexity of the interfaces for locally defined methods and their number are the most important factors for estimating the adaptive maintenance effort.

Acknowledgments

The authors would like to thank the members of MOODS team that have accepted to collect data. A very warm thanks to the several people who have worked for the implementation and test of the several components of our system analizer, TAC++.

References

[1] V. R. Basili, L. Briand, and W. L. Melo. A validation of object oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, pages 751–761, Oct 1996.

[2] P. Bengtsson and J. Bosh. Architecture level prediction of software maintenance. In *Proc. of the 3rd Euromicro Conference on Software Maintenance and Reengineering*, Amsterdam, March 1999. IEEE Press.

[3] G. Booch. *Object-Oriented Design with Applications*. The Benjamin/Cummings Publishing Company, California, USA, 1994.

[4] L. Briand, J. Wurst, S. Ikonomovski, and H. Lounis. A comprehensive investigation of quality factors in object oriented designs: an industrial case study. In *Proc. of the Int. Conf. of Software Engineering (ICSE99)*, LA, USA.

[5] F. BritoeAbreu, M. Goulao, and R. Esteves. Toward the design quality evaluation of object oriented software systems. In *Proc. of 5th International Conference on Software Quality*, Austin, USA, Oct. 1995. McLean.

[6] G. Bucci, F. Fioravanti, P. Nesi, and S. Perlini. Metrics and tool for system assessment. In *Proc. of the IEEE International Conference on Complex Computer Systems*, California, USA, August 1998.

[7] S. N. Cant, B. Henderson-Sellers, and D. R. Jeffery. Application of cognitive complexity metrics to object-oriented programs. *Journal of Object Oriented Programming, JOOP*, pages 52–63, July-Aug. 1994.

[8] M. A. Chaumun, H. Kabaili, R. K. Keller, and F. Lustman. A change impact model for changeability assessment in object oriented software systems. In *Proc. of the 3nd Euromicro Conference on Software Maintenance and Reengineering*, Amsterdam, March 1999. IEEE Press.

[9] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, June 1994.

[10] F. Fioravanti, P. Nesi, and S. Perlini. Assessment of system evolution through characterisation. In *Proc. of the IEEE International Conference on Software Engineering*, pages 456–459, Kyoto, Japan, April 1998.

[11] F. Fioravanti, P. Nesi, and S. Perlini. A tool for process and product assessment of c++ applications. In *Proc. of the 2nd Euromicro Conference on Software Maintenance and Reengineering*, pages 89–95, Florence, Italy, 8-11 March 1998. IEEE Press.

[12] H. M. Halstead. *Elements of Software Science*. Elsevier North Holland, 1977.

[13] B. Henderson-Sellers. Some metrics for object-oriented software engineering. In *Proc. of the International Conference on Technology of Object-Oriented Languages and Systems, TOOLS 6 Pacific 1991*, pages 131–139. TOOLS USA, 1991.

[14] B. Henderson-Sellers. Identifying internal and external characteristics of classes likely to be useful as structural complexity metrics. In D. Patel, Y. Sun, and S. Patel, editors, *Proc. of International Conference on Object Oriented Information Systems, OOIS'94*, pages 227–230, London, Dec. 19-21 1994. Springer Verlag.

[15] B. Henderson-Sellers. *Object Oriented Metrics*. Prentice Hall, New Jersey, 1996.

[16] B. Henderson-Sellers and J. M. Edwards. The object oriented systems life cycle. *Communications of the ACM*, 33(9):143–159, Sept. 1990.

[17] C. F. Kemerer. An empirical validation of software cost estimation models. *Communications of the ACM*, 30(5):416–429, May 1987.

[18] L. A. Laranjeira. Software size estimation of object-oriented systems. *IEEE Transactions on Software Engineering*, 16(5):510–522, 1990.

[19] W. Li and S. Henry. Object-oriented metrics that predict maintainability. *Journal of Systems Software*, 23:111–122, 1993.

[20] M. Lorenz and J. Kidd. *Object-Oriented Software Metrics, A Practical Guide*. PTR Prentice Hall, New Jersey, 1994.

[21] T. J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, 1976.

[22] P. Nesi. Managing oo projects better. *IEEE Software*, pages 12–24, July-Aug 1998.

[23] P. Nesi and M. Campanai. Metric framework for object-oriented real-time systems specification languages. *The Journal of Systems and Software*, 34:43–65, 1996.

[24] P. Nesi and T. Querci. Effort estimation and prediction of object-oriented systems. *The Journal of Systems and Software*, Vol 42:89–102, 1998.

[25] P. J. Rousseeuw and A. M. Leroy. *Robust Regression and Outlier Detection*. John Wiley & Sons, New York, USA, 1987.

[26] O. Signore and M. Loffredo. Some issues ob object-oriented re-engineering. In *Prof. of ERCIM Workshop on Methods and Tools for Software Reuse*, Heraklion, Crete, Greece, 1992.

[27] D. Thomas and I. Jacobson. Managing object-oriented software engineering. In *Tutorial Note, TOOLS'89, International Conference on Technology of Object-Oriented Languages and Systems*, page 52, Paris, France, 13-15 Nov. 1989.