Estimation and Prediction Metrics for Adaptive Maintenance Effort of Object-Oriented Systems

Fabrizio Fioravanti, *Member*, *IEEE*, and Paolo Nesi, *Member*, *IEEE*

Abstract—Many software systems built in recent years have been developed using object-oriented technology and, in some cases, they already need adaptive maintenance in order to satisfy market and customer needs. In most cases, the estimation and prediction of maintenance effort is performed with difficulty due to the lack of metrics and suitable models. In this paper, a model and metrics for estimation/prediction of adaptive maintenance effort are presented and compared with some other solutions taken from the literature. The model proposed can be used as a general approach for adopting well-known metrics (typically used for the estimation of development effort) for the estimation/prediction of adaptive maintenance effort. The model and metrics proposed have been validated against real data by using multilinear regression analysis. The validation has shown that several well-known metrics can be profitably employed for the estimation/prediction of maintenance effort.

Index Terms—Object-oriented metrics, adaptive maintenance effort, estimation/prediction.

1 Introduction

One of the most relevant problems of the maintenance process is the estimation and prediction of related effort [1], [2], [3], [4], [5], [6], [7], [8], [9], [10]. The process of maintenance can be focussed on several different types of interventions on software: correction, adaptation, prevention, etc. For this reason, it is very difficult to define and validate a unique model for estimating effort that applies in all cases. The different maintenance activities may present several related aspects: reuse, understanding, deletion of parts, modifications, development of parts, redocumentation, etc. These problems are relevant for both procedural and object-oriented systems.

Several interesting approaches have been presented for predicting and evaluating maintainability, reusability, reliability, and effort for system development and/or maintenance of object-oriented systems. These high-level features and aspects have been, in many cases, related to complexity or size and sometimes to other aspects concerning system conformity with the object-oriented paradigm. Certain relationships have been demonstrated by using validation processes—e.g., [11], [12], [13], [14], [15]. Classical metrics for procedural languages can be applied with difficulty for the assessment of object-oriented systems [16], [17], [11], [18], [19], [20].

To this end, for the assessment of object-oriented systems, it is very important to take into account the relationships of *is-part-of*, *is-referred-by*, and *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*, also called *has-a by value*, relationships are very important for considering the system composition/decomposition. The *is-referred-by*, also called *has-a by reference* or association, relationships frequently hide dynamic links due to pointers between two or more objects for managing data structures such as lists, trees, and graphs. The *is-a* relationships are a reuse mechanism to exploit polymorphism features of the object-oriented paradigm, but have to be carefully used since inheritance can produce degenerative conditions in which the presence of specialization can also decrease system reusability and maintainability. Object-oriented modeling has been largely adopted in industry in recent years. Unfortunately, systems built four or five years ago may need adaptive maintenance in order to better satisfy market and customer needs and, in some cases, to follow the technology evolution.

In order to control development, as well as maintenance and reengineering, quantitative metrics for evaluating and predicting system characteristics must be used. Effort is one of the most important issues that should be maintained under control (i.e., person-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 or nonlinear relationship between software complexity/size and effort is commonly assumed [21]. Therefore, the problem of effort evaluation is typically shifted to a complexity or size evaluation process. It should be noted that, when software complexity evaluation is performed after system building, it can be useful for:

1. predicting maintenance costs,
2. comparing productivity and costs among different projects,
3. assessing development process efficiency and parameters, and
4. predicting reengineering costs.

• *The authors are with the Department of Systems and Informatics, University of Florence, Via di S. Marta 3, 50139, Florence Italy. E-mail: {fioravanti@dsi, nesi@ingfi1}.unifi.it.*

Manuscript received 1 Feb. 2000; revised 21 July 2000; accepted 1 Dec. 2000. Recommended for acceptance by S. Pfleeger.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 111365.

When software complexity evaluation is performed before system building, it can be used for predicting development, testing, early maintenance costs, etc.

Moreover, on the basis of knowledge that is present in the early stages of the software life-cycle (e.g., number of classes, main class relationships, number of methods, method interfaces, etc.), the process of system analysis allows for the definition and tuning of metrics for predicting effort. From the cognitive point of view, the observable complexity can be regarded as the effort required to understand subsystem/class behavior and functionalities. This complexity can usually be evaluated in the early phases and can be used for predicting costs of reuse and maintenance [22] or for estimating and predicting other features—e.g., [11], [23], [24], [25], [14], [26], [17], [27], [28].

In this paper, a study of the estimation and prediction of adaptive maintenance effort for object-oriented systems coded in C++ is presented. As a result, a model and metrics for effort estimation/prediction of adaptive maintenance are proposed and compared with traditional models and metrics taken from the literature. For the proposed model, new metrics and some well-known metrics are adopted and validated. The validation has been performed by using real data coming from a multipartner project. The validation presented shows that several metrics that can be profitably employed for effort estimation/prediction can also be successfully used for estimation/prediction of adaptive maintenance effort by using the proposed model. The metrics presented in this paper are an extension of a framework specifically defined for the C++ language [17], [11], [29], [30], [31], [32].

This paper is organized as follows: In Section 2, the metrics proposed for evaluating and predicting effort for the adaptive maintenance of each class on the basis of complexity/size are reported. In Section 3, the validation of the most important metrics proposed is presented together with a comparison of metrics extracted from the literature. A descriptive statistic of metrics considered is also reported. The validation proposes a study about the evolution of metric parameters in the adaptive maintenance process. The evolution of metric parameters can be profitably used to deduce/study the trend of other projects. The metrics considered have been validated against real data by using a multilinear regression analysis. The validation shows the relevance of metric terms during adaptive maintenance, considering the differences with other estimation models. Conclusions are drawn in Section 4.

To help the reader understand the metric formulation and discussion, the authors have prepared the glossary reported at the end of the paper, Table 23, in which the metrics and their corresponding meaning are listed in alphabetic order.

2 Metrics for Estimation/Prediction of Adaptive Maintenance Effort

In this section, a new model and metrics for the estimation and prediction of the adaptive maintenance effort of object-oriented systems are presented. The model is based on classical metrics for effort estimation of object-oriented systems development. To this end, a selection of metrics from the literature has been performed on the basis of the authors' experience (e.g., [17], [11], [29], [31], [30], [21]) and considering the several similar experiences presented in the literature—e.g., [28], [12], [25], [33], [16], [34], [14], [35], [36] and [37].

A validation for the proposed metrics for prediction/estimation for the adaptive maintenance effort is reported in Section 3. The validation process has been performed by using multilinear regression analysis and other techniques [38].

Before presenting the new model/metric, some already known metrics are presented since the model is based on these.

2.1 Class Complexity and Size

At the method level, traditional functional metrics such as McCabe's Cyclomatic Complexity, Vg', [39], [40], the Halstead measure, Ha [41], and the number of lines of code, LOC, can be used. These metrics are not very suitable for assessing object-oriented projects since they are not capable of taking into account all the object-oriented aspects [23], [24], [11]. In fact, they neglect information about class specialization (*is_a*: code and structure reuse) and class aggregation and association (*is_part_of* and *is_referred_by*: class/system structure definition and dynamic managing of object sets, respectively). Metrics WMC (Weighted Methods for Class) [25] and LOC (used in [28]) have been adopted as good compromises between precision and simplicity of evaluation for measuring the development effort.

In [31], [11], 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. Class complexity/size, CC, has been defined as the sum of the External Class Description (ECD) and the Internal Class Implementation (ICI):

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

where ECD is the complexity/size due to class definition including method interface definition, while ICI is the complexity/size due to method implementation. The above mentioned CC components can be decomposed in complexities due to local and inherited class members:

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

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

where $ECDL$ is External Class Description Local, CL is the Class complexity/size due to Local methods, $ECDI$ is External Class Description Inherited, and CI is the Class complexity/size due to Inherited methods. Metrics $ECDL$ and $ECDI$ are defined as follow:

$$ECDL = w_{CACL}CACL + w_{CMICL}CMICL,$$
$$ECDI = w_{CACI}CACI + w_{CMICI}CMICI,$$

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

Metrics $CACL$ and $CACI$ are estimated by considering the complexity/size (CC) of class attributes involved. Metrics $CMICL$ and $CMICI$ are estimated by considering the complexity/size (CC) of class method parameters. Metrics CL and CI measure the complexity/size of the functional part estimated by using one of the above mentioned metrics: Ha, Vg', and LOC. Thus, according to the metric used, CC may be considered a complexity or a size metric [42], in which details about the estimation of metric terms and their formal properties are reported. Similar results can be obtained by using Vg', while with Ha the metric is less precise in estimating and predicting effort [11].

Metric CC takes into account:

1. structural aspects (attributes, relationships of *is_part_of* and *is_referred_by*),
2. cognitive aspects (methods, method "cohesion" by means of $CMICL$ and $CMICI$, respectively), and
3. functional aspects by means of CL and CI.

The class attributes can be class instances (evaluated by considering metric CC of their corresponding class), or basic types (e.g., `char`, `int`, `float`, etc.) for which the complexity is posed to predefined values according to the functional metric used.

In the rest of the paper, CC has always been estimated by using LOC as the basic functional metric.

Weights in the above metrics have to be evaluated by using multilinear regression [38] on the basis of the actual class effort [11]. The values of weights obviously depend on the purpose for which metric is used. Thus, a trend for the weights during the development and/or maintenance and/or the reuse process has to be determined. For system development, w_{CACI} and w_{CI} are typically negative stating that the inheritance of attributes and methods leads to savings in complexity/size and, thus, in effort.

In the context of estimating code metrics, the main difference between *predictive* and a posteriori metrics is the consideration or not of the functional code in their estimation. Predictive metrics are estimated by using only the class interface (that is the class/data definition). The class definition is available from the early phases of detailed analysis in which the final cost can be predicted. A posteriori metrics also need class implementations, in terms of method coding. Predictive metrics can be used even in the presence of method implementations; in these cases, they typically give less precise estimations than a posteriori metrics. In general, a posteriori metrics consider all the class aspects: attributes, method interface, and method implementation (both locally defined and inherited). Predictive metrics can be also evaluated if the implementation phase has not yet been performed, such as in the early phases of system development.

The External Class Description metric, ECD, gives a measure of what can be observed by a programmer via the class definition, for example, in the phase of class reuse or maintenance. ECD can be regarded as a particular case of CC when the functional part is not considered or when it is not yet available—for instance, during the early phases of system development or in the use of a library. For this reason, metric CC can also be used for predicting class complexity/size. In particular, the prediction is performed by only considering the class definition: attribute declarations and method prototypes. This estimation can be performed during system analysis/early-design, for example, from the information available in UML class diagrams. The predictive version of the CC metric has the following form:

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

where $CACI'$ and $CACL'$ are estimated on the basis of CC' of class members and, thus, are different from the CC terms. Metric CC' can be based on LOC or token-based metrics. In the rest of the paper, a LOC-based CC' metric has been considered. Even in this case, the weights must be evaluated by using a validation process such as that reported in the next sections.

2.2 Class Attributes and Methods

A lighter approach for class size evaluation can be based simply on counting the Number of Attributes Locally defined (NAL) and the Number of Methods Locally defined (NML) (see also metric $Size2 = NAL + NML$ defined by Li and Henry in [14], which is the sum of the number of local attributes and methods). The counting of class members could in many cases be a too coarse measure. For example, when a class attribute is an instance of a very complex class, it often implies a high cost of method development, which is not simply taken into account by increasing the 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 ($NAML$) and inherited ($NAMI$), respectively:

$$NAM = NAML + NAMI, \tag{5}$$

therefore, NAM can be expanded assuming the form:

$$\begin{aligned} NAM = {} & w_{NAL}NAL + w_{NML}NML \\ & + w_{NAI}NAI + w_{NMI}NMI, \end{aligned} \tag{6}$$

where NMI is the number of inherited methods and NAI is the number of inherited attributes. The typical values of weights can be estimated by using a multilinear regression technique. 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 Related Metrics

The CC metric and its components are related to other metrics taken from the literature: the metric proposed by Thomas and Jacobson [23], that proposed by Henderson-Sellers [24], and the Weighted Method per Classes metric, WMC, of Chidamber and Kemerer [25], [43]. These metrics have been defined as the weighted sum of the number of class members. The suggested weights were numbers

```
class A
{
 private:
		int i;
 protected
		float f;
 public:
		A()
		int get_i();
		float get_f();
		void set_i(int);
		void get_f(float);
};

class B
{
		protected:
				A a;
		public:
				B()
				void set_a(int, float);
};

class C : public B
{
		protected:
				A a1;
		public:
				C();
				void set_a1(int, float);
};
```

```
A::A()
{
		i=0;
		f=0;
}

int A::get_i()
{
		return i;
}

float A::get_f()
{
		return f;
}

void A::set_i(int integer)
{
		i=integer;
}

void A::get_f(float floatingpoint)
{
		f=floatingpoint;
}
```

```
B::B()
{
		a.set_i(5);
		a.set_f(1.5);
}

void B::set_a(int i, float f)
{
		a.set_i(i);
		a.set_f(f);
}


C::C()
{
		a1.set_i(56);
		a1.set_f(15.6);
}

void set_a1(int i, float f)
{
		a1.set_i(i);
		a1.set_f(f);
}
```

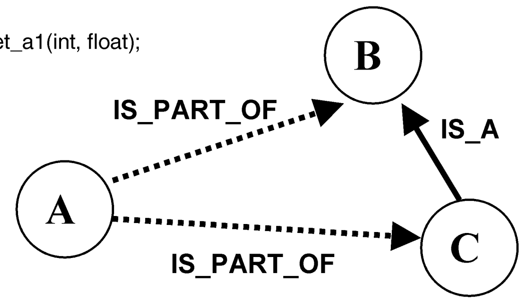


Fig. 1. Example used for the estimation of the cited metrics.

determined on the basis of previous experiences. In some cases, the weight estimation has been performed by using functional metrics [11]. Behind these definitions lies the identification of the terms that can be relevant for the estimation of class complexity/size. Therefore, considering the terms suggested by these metrics and according to CC and NAM, we have obtained the following metrics:

$$TJCC = w''_{CACL}CACL'' + w''_{CL}CL,$$

$$HSCC = w'''_{CACL}CACL''' + w'''_{CL}CL + w'''_{CI}CI,$$

$$TJNAM = w''_{NAL}NAL + w''_{NML}NML.$$

In these metrics, the weights are different from those used for CC or NAM since they are estimated according to a different definition of the class complexity/size metric. For the same reason, $CACL''$ and $CACL'''$ are different from the corresponding terms used in CC or CC' since they are based on a different estimation of class level complexity—e.g., $CACL''$ is the sum of the class complexity of the attributes it estimated by using $TJCC$. For these reasons, weights are distinguished from the previous by using double and triple primes. Metrics CL and CI are based on the simple estimation of functional metrics independently from the class complexity estimation. Metrics NAL and NML are based on counting class members. In addition to these metrics, WMC can also be considered for its wide diffusion. This metric is mainly considered as the number of local methods: $WMC = NML$, while, in some cases, is estimated considering a functional metric for weighting class methods. If LOC is considered:

$$WMC(c) = \sum_{m=1}^{NML(c)} LOC(m) = CL, \qquad (7)$$

where m is the generic method of class c. In the rest of the paper, WMC will be considered in this more complex version, while, in its simpler version, it is discussed as NML. Note that, according to the definition of these metrics, CC and $NAML$ metrics are a generalization of most of the metrics proposed in the literature.

2.4 A Reference Example for the Defined Metrics

In Fig. 1, a small C++ example is reported in order to better explain the mechanisms used for the estimation of metrics reported in the previous sections. The class diagram of the presented system is also reported. Note that class C has been derived from class B and both these classes include an attribute of class A.

The values of the basic metrics discussed before are reported in Table 1; while, in Table 2, some of the

TABLE 1
Basic Metrics Calculated on the Reference Example of Fig. 1

Metric	Class A	Class B	Class C
NAL	2	1	1
NAI	0	0	1
NML	4	1	2
NMI	0	0	1
$CACL$	2	23	23
$CACI$	0	0	23
$CMICL$	5	3	3
$CMICI$	0	0	3
CL	16	8	4
CI	0	0	8
$CACL'$	2	7	7
$CACI'$	0	0	7
$CMICL'$	5	3	3
$CMICI'$	0	0	3

composite metrics that have been previously cited are calculated by considering their weights to be equal to 1. In this case, LOC metric has been used for the estimation of CC. Note the nonlinear behavior of CC with respect to the LOC of the example and the different behavior of metrics NAM and $Size2$ with respect to CC and CC' in assessing class B. For NAM and $Size2$, class B has the lowest values compared to those of the other classes, while the CC and CC' metrics produce an intermediate value. This is due to the fact that the NAM and $Size2$ metrics do not take into account the complexity/size of attributes and methods in their estimation.

2.5 System Level Metrics

The measurements performed on system components/classes are composed to obtain the measure of the whole system. Even in this case, compositional operators are used. An object-oriented metric X is calculated at the system level according to the following equation:

$$S_X = \sum_{i=1}^{NCL(S)} X(c_i), \tag{8}$$

where X is the class-level metric used, NCL is the number of system classes, and c_i is the ith class of system S. This approach can be used for estimating the following system level metrics: S_{CC}, $S_{CC'}$, S_{NAM}, etc.

2.6 Metrics for Adaptive Maintenance Effort

As demonstrated in [11], some of the above-mentioned metrics for complexity/size estimation are strongly correlated with the development effort. In this section, a new model and metric for the estimation and prediction of class effort for adaptive maintenance is proposed. The model can be applied by using the above-mentioned metrics.

The system/class effort for adaptive maintenance is typically spent performing several operations: comprehension/understanding, addition and/or deletion of parts, and modifications/changes of other system code portions. The resulting model is:

$$Eff_{am} = Eff_{add} + Eff_{und} + Eff_{del} + Eff_{chang},$$

TABLE 2
Composite Metrics Calculated on the Reference Example, by Using Weights Equal to 1, of Fig. 2

$Metric$	Class A	Class B	Class C
NAM	6	2	5
$Size_2$	6	2	3
CC	23	34	64
CC'	7	10	20

where Eff_{am} is the effort of class/system adaptive maintenance, Eff_{add} the effort due to the addition of new system parts, Eff_{und} the effort for system/class understanding; Eff_{del} the effort for deleting parts and Eff_{chang} the effort for changing/modifying parts. Note that the understanding effort, Eff_{und}, is the first step for operating changes: deletion, reuse, and modifications. This is a more general model with respect to that used in [5]. The term Eff_{chang} can be eliminated since the changes can be regarded as decomposed into the phases of deleting and of rewriting (adding) of specific parts. In this way, the model is reduced to three terms:

$$Eff_{am} = Eff_{add} + Eff_{und} + Eff_{del}.$$

If the intention is to estimate/predict the maintenance effort on the basis of the available data, the above-considered terms for the maintenance effort have to be estimated by using specific indirect metrics. These metrics have to be based on code analysis, considering both the system/class code *after* and *before* the adaptive maintenance process. If the estimation is performed at the class level, then complexity/size metrics such as CC, CC', WMC, $TJNAM$, $Size2$, and NAM can be used, while the corresponding S_X metrics can be used at the system level. In Fig. 2, the relationships between system/class after and before the adaptive maintenance are depicted.

An estimation of the effort related to the added parts, Eff_{add}, can be performed by using:

$$Eff_{add} \approx M_a - M_r,$$

where M_a is a system/class measure of effort performed on the code *after* the activity of adaptive maintenance and M_r is a measure obtained on the *reused code* of the system/class *before* the adaptive maintenance.

Eff_{und} is the effort used to understand the system and to decide the actions needed for the adaptation. It is reasonable to suppose that Eff_{und} is related to the class/system measure before the adaptive maintenance. For this reason, the effort of understanding can be approximately estimated by using $Eff_{und} \approx k_{und} M_b$, considering effort as directly related to the effort for developing the size/complexity of system/class before the adaptive maintenance process, and k_{und} specifies the relationship (as a scale factor) between effort of understanding and the effort used for producing the code *before* the adaptation process. This scale factor should be determined during the validation process. If the team, comprised of programmers and managers, which performs the adaptive maintenance is the same as that which has created the application before the adaptation,

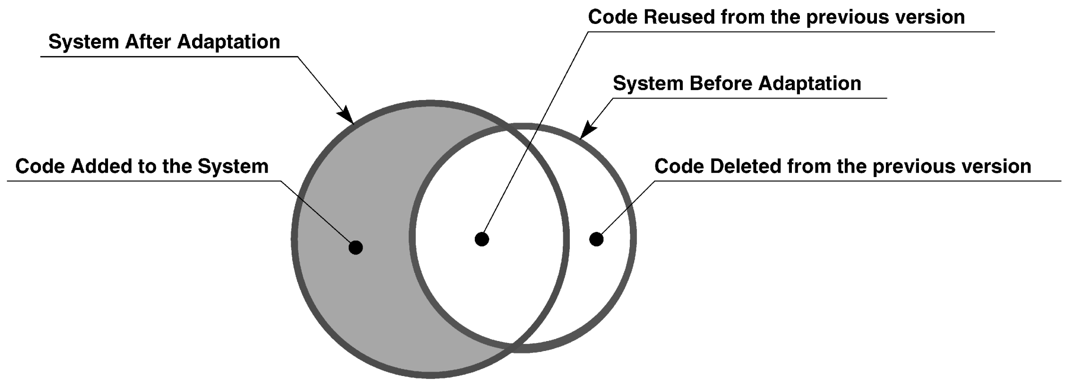

Code Reused from the previous version
System After Adaptation
System Before Adaptation
Code Added to the System
Code Deleted from the previous version

Fig. 2. Relationships among the system after and before the maintenance process.

then the effort for comprehending the system can be neglected with respect to the other factors. Under these conditions, the effort is mainly due to the addition and deletion of code pieces and, thus, $Eff_{und} \approx 0$.

In most cases, Eff_{del} may be very difficult to directly estimate by analyzing the system code. If a LOC-based metric is used, the counting of removed lines of code can be an approximated mechanism to perform its estimation. Unfortunately, this approach cannot be used when more complete and complex metrics are used (such as CC) since structural and functional aspects and the class definitions and relationships have to be considered. Typically, the programmers are quite reluctant to delete methods since, in many cases, it may be difficult and time consuming to be sure that they are not used in other classes or by other team members. Thus, code deletion in classes is typically limited to parts of methods and only in some cases to entire methods or attributes. This produces a well-known maintenance problem for object-oriented systems: the presence of nonused methods and attributes. In general, Eff_{del} can be approximately estimated by using:

$$Eff_{del} \approx k_{del} M_{del} = k_{del}(M_b - M_r),$$

where M_{del} is a system/class measure of code deleted from the *before* version of the system during the activity of adaptive maintenance, k_{del} specifies the relationship (as a scale factor) between deleting effort and the measure of the deleted code, and M_b is a system/class measure of code *before* the adaptive maintenance. The effort for deleting is considered proportional to the estimated size/complexity of the system/class code before the adaptive maintenance process, minus the reused parts of the same code. The scale factor is needed since the activity of deletion has a different effort cost with respect to either the adding or understanding of parts. The scale factor should be estimated by means of the validation process. Typically, the deleted parts are only a limited percentage of the system/class before the maintenance process. For these reasons, the model assumes the form:

$$Eff_{am} \approx M_a - M_r + k_{und} M_b + k_{del}(M_b - M_r).$$

In this equation, the values of metrics can be obtained on the basis of the code if the reused portion of the *before* system is known. The estimation of the unknown parameters k_{und} and k_{del} could be performed by a validation process based on a multilinear regression technique.

This model can be simplified by considering that the reused parts can be represented as a percentage of the system/class before the adaptive maintenance process $M_r = k_b M_b$, then

$$Eff_{am} \approx M_a - k_b M_b + k_{und} M_b + k_{del} M_b - k_{del} k_b M_b.$$

This can be rewritten obtaining the model

$$Eff_{am} \approx M_{am} = M_a - w_{M_b} M_b, \qquad (9)$$

where M_{am} is a metric for effort estimation during adaptive maintenance, typically related to effort, while the weight w_{M_b} is defined as $w_{M_b} = +k_b - k_{und} - k_{del} + k_{del} k_b$. The estimation of each weight is impossible since they are all multiplied by the same factor; therefore, they are included in w_{M_b} as a unique weight. Weight w_{M_b} can be estimated by means of a validation process with a linear regression such as that reported in Section 3. In addition, M metrics may have internal weights that may be influenced by the adaptation process performed as discussed in the next sections—see, for example, CC, CC', $TJCC$, $HSCC$, and NAM.

The above model takes into account:

1. the addition of new classes or parts of them for addressing new functionalities;
2. the deletion of classes or parts of them in the *before* version;
3. the changes in the *before* version;
4. the reuse of classes or portions of them from the *before* version.

The model of (9) can be directly used at the system level. If the model is used at the class level, the basic costs of addition, deletion, and understanding are referred to each class. This means that (9) can be used assuming a unique value for weight w_{M_b}. This implies that changes for the adaptive maintenance are supposed to be uniformly performed on all system classes, while during the adaptive maintenance several distinct actions may be performed in a nonuniform manner.

At class level, the uniform adaptation can be a correct model depending on the adaptation process performed. For example, if the adaptation consists of porting a software application from one operating system to another, the changes could be uniformly distributed in some classes and totally missing in others. If the hypothesis of uniformity is not verified for all system classes, it can be satisfied at the subsystem level or at level of clusters of classes [21]. In this way, the limitation is removed by simply partitioning the system into clusters. The above model remains valid by estimating different values of weight w_{M_b} for each cluster of classes that suffered similar transformations during the adaptive maintenance process.

In any case, the structure of the empirical relationship between effort and measure holds the structure of (9).

2.6.1 CC-Based *Metrics for Adaptive Maintenance*

Considering $M \equiv CC$ in (9), the effort spent during the adaptive maintenance can be expressed as

$$CC_{am} = CC_a - w_{CC_b} CC_b^*,$$

then w_{CC_b} can be included in the weights of CC_b, thus $CC_b = w_{CC_b} CC_b^*$:

$$\begin{aligned} CC_{am} = CC_a - CC_b = {} & w_{CL_a} CL_a + w_{CI_a} CI_a \\ & + w_{CACL_a} CACL_a + w_{CACI_a} CACI_a \\ & + w_{CMICL_a} CMICL_a + w_{CMICI_a} CMICI_a \\ & - w_{CL_b} CL_b - w_{CI_b} CI_b - w_{CACL_b} CACL_b \\ & - w_{CACI_b} CACI_b - w_{CMICL_b} CMICL_b \\ & - w_{CMICI_b} CMICI_b. \end{aligned}$$

Since CC is defined as the weighted sum of six terms, the whole metric CC_{am} has 12 terms and their corresponding weights, six for CC_a, and six for CC_b. A complete validation of the CC_{am} metric has to be performed by considering the whole structure of CC. In that case, a multilinear regression can be used to estimate suitable weights. This validation also allows for identifying terms of metric CC_{am} that are relevant for effort estimation of the adaptive maintenance. Metric CC_b also takes into account some cognitive aspects of the system; for example, it partially models the effort in comprehending the system before adaptation. In the past, other linear and nonlinear models have been considered [5], [7], [8], [9]. The adoption of a linear or nonlinear model depends on the structure of the basic metric, M, used for the definition of the maintenance metric. Typically, the basic metrics are selected from those which are strongly related to the development effort. For the same reasons, the CC metric [11] was selected. Note that the CC metric considers the nonlinearity of effort since CC terms take into account both structural and functional aspects iteratively along the class hierarchy [11].

A predictive version of CC_{am} can be obtained by considering $M \equiv CC'$ in (9):

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

In this case, weight $w_{CC'_b}$ can also be included in the terms of CC'_b ($CC'_b = w_{CC'_b} CC'^*_b$); then, a multilinear regression has to be used for estimating the eight weights of CC'_{am}. Once the system analysis of the adaptation phase is performed, the structures that system classes will have after the adaptive maintenance are known. With this knowledge, it is possible to use the CC'_{am} metric for predicting the adaptive maintenance effort.

2.6.2 NAM-Based *Metric for Adaptive Maintenance*

Considering $M \equiv NAM$ in (9), effort spent for the adaptive maintenance can be expressed as

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

Weight w_{NAM_b} can be included in the terms of NAM_b (therefore, $NAM_b = w_{NAM_b} NAM_b^*$). The validation has to be performed by considering the whole structure of NAM for both NAM_a and NAM_b and, thus, estimating eight weights. After a draft analysis of the adaptation phase that has to be performed, the number of members that each class will present at the end of the adaptive maintenance process is known. Therefore, by using the NAM_{am} metric, it is possible to use this early knowledge for predicting the adaptive maintenance effort.

2.7 Other Metrics and Models for Adaptive Maintenance

The above approach can be used for defining general metrics for the estimation of adaptive maintenance effort, on the basis of the above-mentioned metrics: $HSCC_{am}$, $TJCC_{am}$, and $TJNAM_{am}$. For the WMC and $Size2$ metrics, a different approach has to be used since they do not include weights in their definition. For this reason, in order to estimate the model by applying techniques reported in [38] for $Size2$ or WMC metrics, it is necessary to adjust the model according to the following equation. For example, for $Size2$:

$$Eff_{am} \approx Size2_{am} = Size2_a - w_{Size2_b} Size2_b,$$

and then,

$$Size2_a - Eff_{am} \approx w_{Size2_b} Size2_b.$$

Weight w_{Size2_b} is a scale factor that can be estimated with a linear regression. Note that $TJNAM_{am}$ is based on counting the same aspects as for the $Size2_{am}$ metric. The only difference is the number of weights.

For NML, a model with two weights has been used. This model is more precise than the model used for $Size2_{am}$ and WMC_{am} since the general metric can be more precisely tuned:

$$Eff_{am} \approx NML_{am} = w_{NML_a} NML_a - w_{NML_b} NML_b.$$

A very different model for estimating adaptive maintenance effort can be defined by considering only the measure on the code *after*:

$$Eff_{am} \approx M_{am0} = w M_a. \tag{10}$$

Weights w can be included in those of metric M_a, if any. This model can be used for defining several different metrics based on: CC, CC', NAM, $Size2$, etc., obtaining CC_{am0}, CC'_{am0}, NAM_{am0}, $Size2_{am0}$, etc.

Another different model with respect to those previously discussed can be defined by considering the same weights for both *after* and *before* terms:

$$Eff_{am} \approx M_{am1} = \sum_{i=1}^{N} w_i (T_a(i) - T_b(i)), \tag{11}$$

TABLE 3
Summary of Metrics and Models for the Estimation and Prediction of the Adaptive Maintenance

	Measuring models			*Metric Description*	
M metric	M_{am}	M_{am0}	M_{am1}	*Type*	*Class Features*
CC	12	6	6	Functional-based	Attributes and methods both locally defined and inherited, plus method interface
$HSCC$	6	3	3	Functional-based	local and inherited methods plus local attributes
$TJCC$	4	2	2	Functional-based	local attributes and methods
WMC	1	1	1	Functional-based	local attributes
CC'	8	4	4	Functional-based	Attributes and methods interfaces both locally defined and inherited
NAM	8	4	4	Counting members	attributes and methods locally defined and inherited
$TJNAM$	4	2	2	Counting members	local attributes and methods
$Size2$	1	1	1	Counting members	local attributes and methods
NML	2	1	1	Counting members	local methods

where, for example, $T_a(2)$ is the second term of metric M_a (after the adaptive maintenance) and w_i is the weight associated with both *after* and *before* i-indexed terms of metric M. This approach can be applied to all the above-mentioned metrics. This model is probably the most used for estimating and predicting the maintenance effort. In this case, the terms of the metrics are previously combined and then the weights are applied. For instance, NAM_{am1} presents only four weights instead of the eight weights of model M_{am} discussed in the previous sections:

$$\begin{aligned} &NAM_{am1} = \\ &w_{NAL_{am1}}(NAL_a - NAL_b) + w_{NAI_{am1}}(NAI_a - NAI_b) \\ &\quad + w_{NML_{am1}}(NML_a - NML_b) + w_{NMI_{am1}}(NMI_a - NMI_b). \end{aligned}$$

The same approach can be used to obtain CC_{am1}, WMC_{am1}, and $Size2_{am1}$, etc.

2.7.1 Metric and Models Summary

In Table 3, a summary of the considered metrics and models is reported. In the first column, the considered metrics are listed, for each metric a short description is included on the last two columns: *type* and *class features*. Column *Type* states the approach for metric estimation: *Functional-based*, when the metric is based on functional metrics such as LOC, Vg', etc.; *Counting members*, when the metric is estimated by counting the specified class members. Column *Class Features* briefly describes the terms/class members considered during the metric estimation. The metrics are ordered from the more complete to the simpler: A metric has been considered more complete than another if it takes into account a greater number of aspects/factors of the object-oriented paradigm (such as class members, relationships) and phases of the maintenance process. For example, metric $Size2$ is less complete than NAM since it does not take into account inherited aspects and the latter metric is less complete than CC for the lack of the functional part.

The internal columns are related to the measuring models described in the previous sections. These columns report the number of weights that are included in the model defined depending on the metric M chosen. The number of weights is related to the number of additive terms of the metric and to the model used; for instance, models $am0$ and $am1$ produce metrics with the same number of weights, while model $am1$ uses two different measures of the system, that is, after and before. As will be shown in the next section, more complete models and metrics lead to more precise results.

In the next section, the adoption of the above presented metrics based on the models considered for estimation/prediction of maintenance effort is discussed by using validation techniques. These metrics can be useful for 1) comparing productivity and costs among different projects and 2) learning the development process efficiency and parameters for further estimation and prediction.

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 object-oriented systems.

The comparative analysis with validation has been carried out among the previously defined metrics and some of those already defined in the literature. Moreover, an analysis to verify the influence of metric parameters in producing the final results has been performed in order to identify the minimum number of parameters that are needed to obtain suitable measures. Therefore, the analysis performed is more than a simple validation. It has produced a clear view of the behavior of the above-mentioned metrics for estimating class effort for adaptive maintenance.

The validation has been performed by considering the data related to the MOODS ESPRIT IV project (Music Object-Oriented Distributed System). The project consisted of the implementation of a distributed system 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 into a distributed system of music editors, for cooperative management of music. The reference TTN was

TETRApc having as partners CPR (Consortium Pisa Research) and CESVIT (High Tech Agency).

MOODS has been a multipartners project evaluated in about 44 man months including hardware and software. Hardware effort was mainly devoted to implementing special music lecterns. Software effort was used to

1. port the stand alone music editor from MS-DOS to UNIX,
2. include the possibility of editing main music scores,
3. adapt the original version of the music editor towards a distributed co-operative version,
4. implement an independent orchestra network configurator and monitor,
5. implement several additional advanced functionalities for the cooperative music editor, and
6. connect the music editor to a database of music scores.

The data and the code used in the following validation are referred to points 1 and 2, which are the adaptive maintenance phases of the project. In the MOODS project, the first work-packages have been the porting and the adaptation to the distributed approach, 1 and 2; then phases 3, 4, and 5 have been performed in parallel. Phases 1 and 2 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 become 133 after performing phases 1 and 2, with 9.6 person-months. This version has been called LIOO5 since several intermediate versions were analyzed.

In the following validation, the effort data was collected by using log files detailing the effort performed in two main categories [21]: the effort for coding and readapting the system and the effort for documenting, general testing, taking high-level decisions, management, etc. In the following, two effort data sets are considered:

Eff_{cm}: effort including code manipulation, i.e., deleting code, adding classes, removing classes, changing classes, single class, and method testing, etc.

Eff_{tot}: effort including the above effort of code manipulation, Eff_{cm}, plus the time spent for designing, documenting, general testing, taking high-level decision, management, etc. All these collateral activities are fundamental to the adaptation process and are differently related to the code complexity/size with respect to the activity on code.

The MOODS project has been built by using object-oriented analysis and design methodologies (i.e., Booch [44]) 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 [11], [29], [31], and [30]).

Before presenting the validation of the proposed metrics, the analysis of the system evolution 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 Table 4, 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 percent in the system size due to the addition of new functionalities. This size increment is located in some new classes and quite uniformly distributed in the reused classes. This can be coarsely observed by comparing the mean values of CC and MCC and the values of MNA and MNM that have not changed their values in a significant way after the adaptation. By the value of NRC, it is evident that six classes (roots or stand-alone) have been added. The added classes in our example are stand-alone classes for representing the music main-score. In LIOO1, some classes having a very high value of CC were identified (e.g., CC about 4,300 units). This condition was corrected during adaptive maintenance in version LIOO5 in which the maximum CC is about 3,200 [30]. A further analysis of system evolution during the adaptive maintenance is reported in the following.

TABLE 4
Overview of the System *before* and *after* the Adaptive Maintenance Process

	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

NCL is the Number of Classes in the system, NRC is the Number of Root Classes in the system, TNM is the Total Number of Methods in the system, $TLOC$ is the Total number of LOC in the system, MCC is the Mean value of CC, MNA is the Mean value of the Number of class Attributes, and MNM is the Mean value of the Number of class Methods.

3.1.1 System Effort and Weights Trends

In Fig. 3, the trend of metrics CC, CC', and NAM for effort prediction/estimation is reported together with the actual cumulative effort evolution. The figure covers the time frame from the phase of detailed analysis to the 90 percent of the project. Note that the above graph reports the evolution of LIOO in the MOODS project including the orchestra network configurator. Note that, during the maintenance process (from version 1 to version 5), the estimation/prediction of system effort has been quite precise with all the considered metrics. More precise figures of this behavior are reported at the end of the validation.

Another interesting trend analysis is that of metric weights. In Fig. 4, the trend for weights of CC along the project evolution is reported. The weights have been estimated by means of a multilinear regression analysis during the validation of CC considering the total development effort. This graph refers to the same project and temporal window discussed in Fig. 3. The project presents from phase 1 to 5, the period in which the porting and adaptive maintenance have been performed. LIOO5 is the phase after which the development restarted and LIOO6

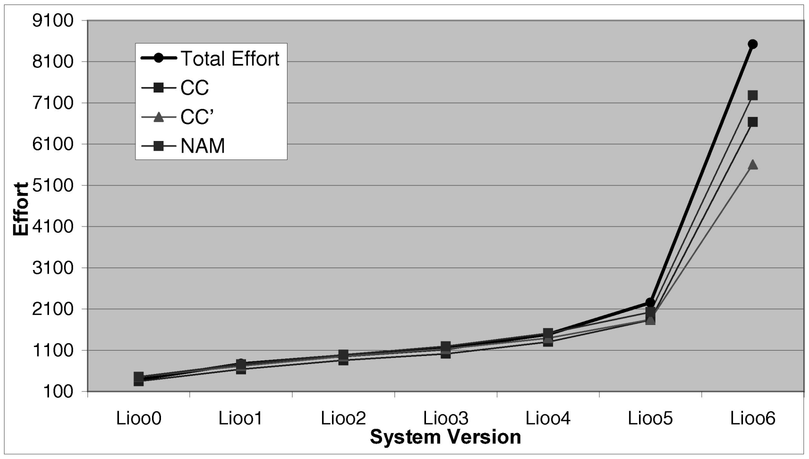


Fig. 3. Trend of the total development effort and the estimated/predicted effort (cumulative) on the basis of CC, CC', and NAM. The trends include LIOO6, which is the first milestone after the adaptive maintenance. It includes the implementation of the orchestra network configurator.

represents the first check point in the development phase after the porting. From Fig. 4, it can be observed that

- w_{CACL} is quite stable, demonstrating that the definition of a local attribute is quite a constant cost during the whole project life-cycle that includes the adaptive maintenance.
- w_{CACI} is negative during the early phases of the software life-cycle (denoting an effort saving). Then, it becomes positive when the software becomes more stable and during the adaptive maintenance. The value of this weight increases during the adaptation. The porting and adaptation have been implemented by restructuring functional parts, including the inherited part, thus the inspection of the attributes of the superclasses has been a cost.
- w_{CMICL} is positive during the project evolution except for LIOO6 for which it becomes negative denoting that an effort saving is possible only after the stabilization of specification and documentation about the methods' interfaces of the classes in the hierarchy.
- w_{CMICI} has quite an opposite trend with respect to w_{CMICL} denoting that a modification of basic class method interfaces in the first phase of the project produces light increments in effort; while a change, when the hierarchy is stable, implies extensive adaptation to all derived classes and, thus, the coefficient becomes positive. During the adaptive maintenance, this weight is negative, stating that the presence of inherited methods was a cost saving. This means that during the porting and adaptation the presence of detailed interfaces for inherited methods was a saving.
- The w_{CL} coefficient is quite stable and always positive for all the project versions denoting that the development of local code is always a cost. The relevance of functional aspects of the classes is constant along the process even during the porting and adaptation.
- The w_{CI} coefficient influences only the last phases of the software life-cycle, in which it helps to save effort (it is negative). When the hierarchy is stable, the code reuse by means of inheritance is always an effort saving.

This example has shown how the trend analysis of weights can give interesting information about the system evolution. In Fig. 5, the trend of weights for CC' metric

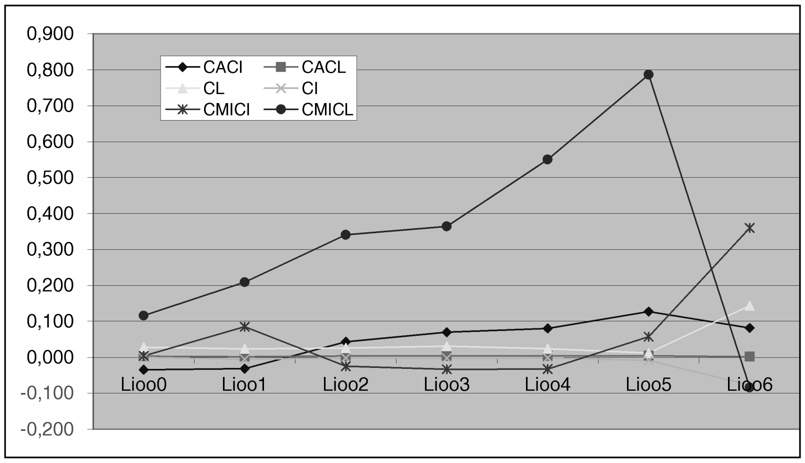


Fig. 4. Trend of CC weights along LIOO project evolution considering total development effort.

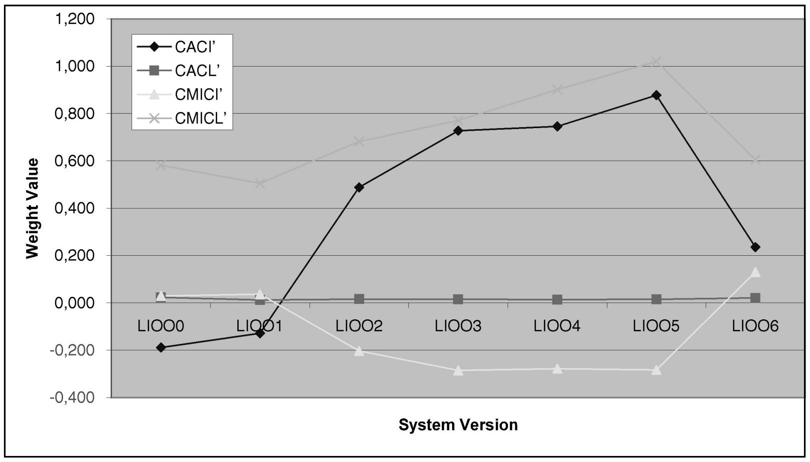


Fig. 5. Trend of CC' weights along LIOO project evolution considering total development effort.

considering the total effort is reported. Their trend is quite similar to that of the corresponding weights of the CC metric. This means that to consider the functional terms does not radically change the behavior of metric terms and of the results produced.

The trends of weights reported in Fig. 4 and Fig. 5 do not change their behavior if the values of weights are normalized with respect to the mean value of their corresponding metrics.

In Fig. 6, the trend for weights of CC along the project development considering the code manipulation effort is reported. From Fig. 6, it can be observed that:

- w_{CACL}, w_{CL}, and w_{CI} denote quite constant costs related to local attributes and methods, both locally implemented and inherited.
- w_{CACI} has a slight increasing value. This can be due to the effort spent in inspecting the class hierarchy in the adaptive maintenance.
- w_{CMICL} and w_{CMICI} have a similar behavior as the corresponding weights for the CC_{am} metric for the total effort (see Fig. 4).

This example has shown how the trend of weights can be used to analyze the system evolution. Moreover, the trend of the weights is quite independent from the effort type considered. As above, the same trend can be recovered by considering the weights of CC' metrics estimated for the development effort.

3.2 Data Analysis

The distribution of changes along system classes is typically nonuniform. As already stated, the problem of nonuniform distribution of changes (addition, deletion, reusage) can be managed by clustering classes with the same history and evolution. The nonuniformity can be easily detected by using tools for tracking changes.

According to the defined model, some other exceptions, that can lead to obtain noncorrect estimations of the adaptive maintenance effort, may exist.

1. *Fusion*. Some classes in the *before* version can be fused into one class in the *after* version. This operation is quite rare but possible during the adaptive maintenance.
2. *Split*. The functionalities managed by a class in the *before* version can be distributed among more classes in the *after* version by moving methods and attributes, and in some cases deleting parts. The

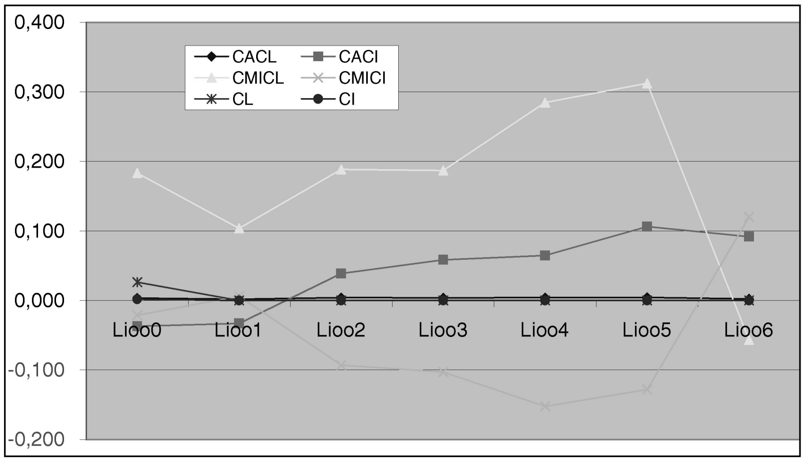


Fig. 6. Trend of CC weights along LIOO project development considering code manipulation effort.

TABLE 5
Classification Analysis of System Changes

Possible System Clusters	number of classes	S_{CC} LIOO5	S_{CC} LIOO1	ΔS_{CC}	% on S_{CC} LIOO5	% on S_{CC} LIOO1	% of Δ on S_{CC}	% Number of classes
Unchanged classes	12	747	747	0	0.9	1.1	0.0	9.1
Completely New classes	21	3685	0	3685	4.6	0.0	30.0	15.9
Split and changed Classes	1 in 2	2746	3640	-894	3.4	5.3	-7.3	1.5
Classes with Changes/additions	98	73705	64208	9497	91.1	93.6	77.3	73.5
Total	133	80883	68595	12288	100	100	100	100

Note the lack of deleted and fused classes, where S_{CC} is the system complexity defined in terms of class complexity, CC, see (8).

split is quite a frequent action when a class becomes too large.

The solution for the case of *Fusion* is to consider the fused classes as a single class since the *before* version. For instance, if classes C_i and C_j of the before version are fused into class C_k in the after version, the following model can be considered:

$$Eff_{am}(C_{ij\rightarrow k}) \approx M_a(C_k) - w_{M_b}(M_b(C_i) + M_b(C_j)).$$

In the same manner, in the case of *Split*, the solution is to consider classes produced by splitting a large class such as a unique class in the metric estimation even in the *after* version:

$$Eff_{am}(C_{k\rightarrow ij}) \approx (M_a(C_i) + M_a(C_j)) - w_{M_b}M_b(C_k).$$

The grouping of classes with similar needs/histories of adaptation is a solution for getting more precise predictions/estimations, even for systems presenting a significant number of instances of these cases. The weights related to each of these groups can be separately considered and estimated via separate validation processes if the number of classes is large enough to perform a regression analysis. If the history of system classes is not known, it is also possible to make an analysis of system evolution by using tools for tracking changes and duplicated code.

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 the MOODS project. The validation process has been performed by using a multilinear least squares regression technique [38] considering 1) the relationship between effort and metrics as linear, 2) the values of direct metrics, and 3) 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 a certain relevance to the validation process.

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 even if the 26 percent of system classes have not suffered a uniform adaptive maintenance.

In Table 5, the main clusters of classes which have been identified on the basis of the changes performed on the system during the adaptive maintenance are reported. The main group is comprised of classes with changes and additions. These classes are responsible for the 91 percent of the whole system complexity. Classes of this group present quite uniform modifications. The system evolution has not produced deleted and/or fused classes. Only one class has been split into two classes and, to these, additional code has been added. The other significant clusters are those related to unchanged classes, completely new classes, and split and changed classes. These clusters present a low number of classes and are only a small part of the whole system complexity/size.

By using real data of adaptive maintenance effort (provided in man/hours), the weights for CC_{am}, CC'_{am}, and NAM_{am} have been calculated by minimizing the least squares error [38]. The correlation values have been reported with all the data evaluated in order to have an immediate figure of the validity of the results obtained.

As already discussed in the previous sections, metrics can be classified as *a-posteriori* and *predictive* metrics. The validation/analysis of metrics for these two categories are separately discussed in the next sections.

The empirical validation proposed includes several aspects and techniques. These contribute to provide a quite comprehensive model for the metrics considered. To this end, a statistical description of metrics has been implemented and, thus, a multilinear regression for the adaptive maintenance effort has been used. In Table 6, the typical values for the above metrics estimated by considering the projects listed in Table 4 have been reported. In the table, the maximum, the median, the minimum values, and the standard deviation are reported for each metric. The values reported in the table for metrics NAL, NML, NAI, NMI, and $Size2$ are in accordance with those obtained by other researchers [28], [16] when guidelines of object-oriented analysis and design are applied by skilled people.

In the next sections, the results of the multilinear regression analysis are given for the adaptive maintenance:

1. a-posteriori estimation of total effort;
2. a-posteriori estimation of code manipulation effort;
3. predictive estimation of total effort;
4. predictive estimation of code manipulation effort.

At the end, a comparison of results is reported.

In the next sections, if not otherwise specified, the data refers to the period between version 1 and version 5 of LIOO system since the maintenance was mainly performed in that period.

3.3 A-Posteriori Estimation of Total Effort of Maintenance

In Table 7, the results of the multilinear regression analysis for the CC_{am} metric are reported. The regression has been carried out by considering the total effort of adaptive maintenance, Eff_{tot}, in person-hours and the CC_{am} metric

TABLE 6
Typical Value of the Metrics Considered (Descriptive Statistics)

Metric	Max	75%	Median	25%	Min	Mean	StdDev
CL	1243	72	25	14	0	101.04	213.88
CI	1511	134	101	92	0	191.91	289.32
$CACL$	11500	3	0	0	0	257.74	1406.59
$CACI$	405	21	19	19	0	20.98	35.78
$CMICL$	71	14	10	6	0	11.42	11.67
$CMICI$	113	28	18	14	0	25.07	24.40
$Size2$	78	11	7	5	0	11.75	14.48
NAL	17	2	0	0	0	1.92	3.51
NAI	33	5	3	3	0	6.05	7.80
NML	67	11	7	4	0	9.83	11.63
NMI	141	30	25	20	0	29.52	27.97
Eff_{cm}	40.95	2	0	0	0	3.14	7.57
Eff_{tot}	80.94	9.77	6.44	6.44	6.44	11.18	11.26

The values have been obtained considering LIOO5 version. WMC is present in both forms: CL and NML.

with 12 components/weights, by using the techniques discussed in [38]. In Table 7, statistical values showing the confidence of the weights are reported for each component of CC_{am}. Relevant metric terms 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 terms to be estimated, and α is the percentage of confidence (i.e., for α equal to 0.05, we have a confidence interval of 95 percent). In order to avoid the recovering of data from probability tables, *p-values* are also reported. *p-values* represent the probability that a *Student-t* with $n-p$ degrees of freedom becomes larger in absolute value than the corresponding *t-value*. Below, the rows containing the weights and the correlation between the real adaptive maintenance effort and the estimated CC_{am} with the identified weights 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 the *t-value* has to be greater than 1.96 for obtaining a confidence interval of 95 percent. According to the above considerations, the weights of $CACL$ and $CACI$ can be imposed to zero in the CC_{am} structure because both after and before components satisfy the null hypothesis. By imposing a null value on these weights the correlation remains quite stable, 0.885, while, by eliminating any other couple of terms, the correlation decreases significantly.

In the following tables, other results obtained by eliminating *couples* of weights are reported. In general, it is necessary to proceed to eliminate or consider both terms (*after* and *before*) valid; otherwise, the M_{am} metric could partially lose its symmetric structure (see, in the following, the comparison with other models).

The model presented for the estimation of adaptive maintenance takes into account the weighted difference between corresponding terms (e.g., $w_{CMICL_a}CMICL_a - w_{CMICL_b}CMICL_b$), the weight sign must be carefully considered. For this reason, a negative term represents an effort saving, while a positive term represents a cost. In the proposed model, the signs of weights are equal since they are applied to the metric *before* or *after* the adaptation process. This is due to the presence of the minus sign in the proposed model (see (9)). According to Table 5, different signs can be found for the metrics terms. Typically, the terms *after* have higher values than their corresponding *before* terms (that is, $T_a \geq T_b$). This is due to the fact that classes after the adaptation typically present an increment in size/complexity. For this reason, considering a couple of corresponding terms we have two cases if $T_a \geq T_b$.

- Case 1. If $w_a > w_b > 0$, then $w_aT_a - w_bT_b > 0$ and, thus, the corresponding term is a cost.
- Case 2. If $w_a < w_b < 0$, then $w_aT_a - w_bT_b < 0$ and, thus, the corresponding term is a saving.

If $w_a < w_b$ and the weights are positive or if $w_a > w_b$ and the weights are negative, nothing can be said about the cost or saving for the terms, in general. A more accurate analysis could be performed only by observing the single class or cluster. Therefore, the above cases can be applied for systems and clusters in which the deleted portions can be neglected with respect to the additions performed.

TABLE 7
Results of the Multilinear Regression Analysis of CC_{am} Metric for Total Effort Evaluation, Eff_{tot}

$CC_{am} \approx Eff_{tot}$			
	w	$\mid t-value \mid$	$p-value$
$CACI_b$	-0.486	1.866	0.064
$CACL_b$	-0.004	0.764	0.447
CL_b	-0.010	1.459	0.147
CI_b	-0.072	2.669	0.009
$CMICI_b$	0.743	2.531	0.013
$CMICL_b$	2.451	12.819	0.000
$CACI_a$	-0.286	1.287	0.201
$CACL_a$	-0.003	0.596	0.553
CL_a	-0.024	2.925	0.004
CI_a	-0.066	2.934	0.004
$CMICI_a$	0.766	3.009	0.003
$CMICL_a$	2.681	13.559	0.000
R-squared	0.771		
F-stat (p-val)	33.760 (0.000)		
Correlation	0.899 with all components		
	0.885 by removing $CACL$ and $CACI$		
Std.Dev.	11.504 with all components		
	10.778 by removing $CACL$ and $CACI$		

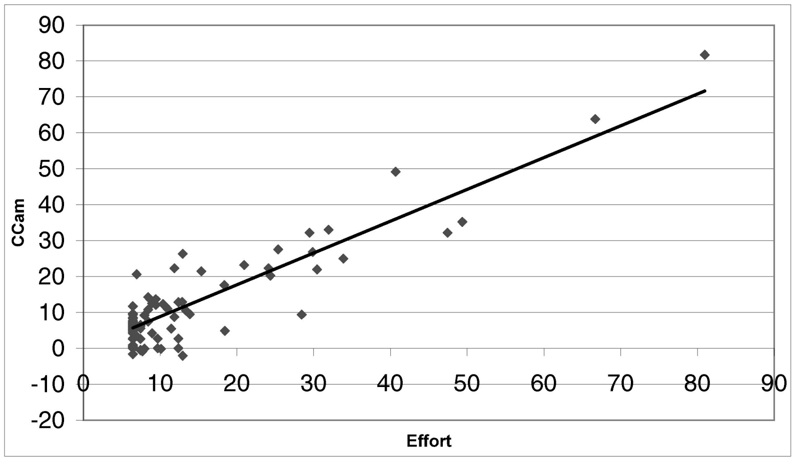


Fig. 7. Dot diagram representing the correlation between CC_{am} and Eff_{tot}.

In Table 7, it should be noted that the terms related to attributes (both locally defined and inherited) are not relevant from a statistical point of view. Even if they contribute to increase the correlation, their removal does not change a lot the correlation level. According to the previous considerations, the terms that increase the effort are mainly related to the method interface: $CMICL$ and $CMICI$ terms. This behavior is coherent with the process of adaptation in which the inspection of class interfaces is frequent.

In Table 7, the best correlation is obtained by considering valid the values of the coefficients that are not significant from a statistical point of view. In Fig. 7, CC_{am} is depicted by means of a dot diagram showing real values of effort against the estimated ones.

For $TJCC_{am}$, a strong decrement of correlation has been obtained (correlation equal to 0.62, with a Standard Deviation of 10.094). This decrement of correlation, with respect to that of CC_{am}, is mainly due to the lack of inherited terms in $TJCC$. In this model, the term related to class attributes, $CACL$, has no statistical meaning, but its removal decreases the correlation by about 10 percent.

Metric $HSCC$ is obtained by adding to metric $TJCC$ the term related to the inherited code, CI. The corresponding results are reported in Table 8. In this case, the $CACL$ term is not significant according to the multilinear regression analysis. As in the previous examples, the removal of this component decreases the correlation.

Metric WMC_{am} presents a low correlation with the total maintenance effort spent during the adaptation process. This fact is highlighted by a correlation of 0.6 with a standard deviation of 9.5. The estimation of this version of WMC is quite cheap since it consists of the counting of the LOC for methods.

Metrics CC'_{am}, $TJNAM_{am}$, $Size2_{am}$, and NML_{am} are discussed together with prediction metrics in Section 3.5.

According to the above analysis, CC_{am} is a suitable metric for maintenance effort estimation by using the nonstatistically meaningful coefficients, such as highlighted in Section 3.7. The generic model allows estimating of the maintenance effort with a significant accuracy. The standard deviation has to be regarded as a measure of the spreading-out of the estimation and thus also of the feature under estimation. The standard deviation of total Effort, Eff_{tot} (see Table 6), is 11.26, which is comparable to the standard deviations found for the previous metrics.

3.4 A-Posteriori Estimation of Code Manipulation Effort of Maintenance

In Table 9, the results related to the validation of the CC_{am} metric for the estimation of code manipulation effort of maintenance are reported. The correlation between that effort, Eff_{cm}, and CC_{am} is lower than those reported in Table 7, where the total effort was considered. The meaningful terms, from the statistical point of view, are the same as in the analysis performed for the total effort. In general, class attributes seem to have a lower influence on effort estimation. The removal of $CACL$ and $CACI$ components provokes a very small change in the correlation.

Note that, when code manipulation effort is considered, the development of local code, CL, is only marginally significant (see $p-value$ of w_{CL_a}), while aspects considered by $CMICI$ are still a cost.

The same considerations reported for $TJCC_{am}$, in the case of total effort, are also valid for code manipulation

TABLE 8
Results of the Multilinear Regression Analysis of the $HSCC_{am}$ Metric for Total Effort Evaluation of Maintenance, Eff_{tot}

$HSCC_{am} \approx Eff_{tot}$	w	$\lvert t-value \rvert$	$p-value$
$CACL_b$	-0.008	1.888	0.061
CL_b	0.033	2.975	0.004
CI_b	0.088	3.966	0.000
$CACL_a$	-0.004	1.120	0.265
CL_a	0.062	5.981	0.000
CI_a	0.077	4.192	0.000
R-squared	0.116		
F-stat (p-val)	2.749 (0.015)		
Correlation	0.635 with all components		
	0.529 by removing $CACL$		
Std.Dev	10.220 with all components		
	8.321 by removing $CACL$		

TABLE 9
Results of the Multilinear Regression Analysis of the CC_{am} Metric for Code Manipulation Effort Evaluation, Eff_{cm}

$CC_{am} \approx Eff_{cm}$	w	$\mid t-value \mid$	$p-value$
$CACI_b$	-0.093	0.545	0.587
$CACL_b$	-0.002	0.636	0.526
CL_b	0.015	3.334	0.001
CI_b	-0.112	6.341	0.000
$CMICI_b$	0.508	2.625	0.010
$CMICL_b$	1.815	14.414	0.000
$CACI_a$	-0.077	0.523	0.602
$CACL_a$	-0.001	0.350	0.727
CL_a	0.007	1.258	0.211
CI_a	-0.098	6.628	0.000
$CMICI_a$	0.559	3.338	0.001
$CMICL_a$	1.745	13.401	0.000
R-squared	0.781		
F-stat (p-val)	19.215 (0.000)		
Correlation	0.884 with all components		
	0.857 by removing $CACL$ and $CACI$		
Std.Dev	6.776 with all components		
	6.836 by removing $CACL$ and $CACI$		

effort, such as reported in Table 10. The main difference is related to the fact that all coefficients are significant. In particular, the $CACL$ term is significant and the sign of the weights of the *after* terms of the metric suggests that an effort increment is related to this component.

In Table 11, w_{CI} is the only nonsignificant term. By removing the CI term (forcing $w_{CI} = 0$), $HSCC$ assumes the same structure of $TJCC$, but with a higher correlation. This fact suggests that the CI term influences the other terms in metric estimation (see the definition of the CC metric). The significance of the CI terms is not drastically low and, thus, it may be included in the model. For the estimation of code manipulation effort, the $HSCC$ correlation is increased to more than 80 percent. Also in this case, local attributes and methods have generated an increment of effort, confirming the above analysis of the adaptive maintenance process.

In this case, metric WMC_{am} presents a correlation of 0.442 with a standard deviation of 3.98. This is coherent

TABLE 10
Results of the Multilinear Regression Analysis of the $TJCC_{am}$ Metric for Code Manipulation Effort Evaluation, Eff_{cm}

$TJCC_{am} \approx Eff_{cm}$	w	$\mid t-value \mid$	$p-value$
$CACL_b$	0.005	1.822	0.071
CL_b	0.038	5.709	0.000
$CACL_a$	0.006	2.650	0.009
CL_a	0.046	7.637	0.000
R-squared	0.258		
F-stat (p-val)	11.142 (0.000)		
Correlation	0.532		
Std.Dev	4.530		

TABLE 11
Results of the Multilinear Regression Analysis of the $HSCC_{am}$ Metric for Code Manipulation Effort Evaluation, Eff_{cm}

$HSCC_{am} \approx Eff_{cm}$	w	$\mid t-value \mid$	$p-value$
$CACL_b$	0.005	1.783	0.077
CL_b	0.036	5.375	0.000
CI_b	0.026	1.914	0.058
$CACL_a$	0.006	2.504	0.014
CL_a	0.047	7.372	0.000
CI_a	0.019	1.755	0.082
R-squared	0.282		
F-stat (p-val)	8.262 (0.000)		
Correlation	0.823 with all components		
	0.802 by removing CI		
Std.Dev	4.668 with all components		
	4.620 by removing CI		

with the behavior of this metric for the total effort estimation. Note that this version of WMC is in practice totally procedural since it consists of counting the LOC. For a different version of WMC_{am}, follow the behavior of NML_{am} which is discussed together with prediction metrics in Section 3.6.

3.5 Predictive Estimation of Total Effort of Maintenance

In this section, some metrics for effort prediction (in absence of terms related to functional code complexity) are reported. In particular, metrics CC', $TJNAM$, NAM, and $Size2$ are applied in the generic maintenance model of (9).

In Table 12, the results obtained for metric CC'_{am} considering total effort, Eff_{tot}, are reported. The correlation obtained for CC'_{am} is very close to that obtained for CC_{am}. Analyzing the statistical values related to the metric terms, it is evident that $CACI'_{am}$ and $CMICI'_{am}$ satisfy the null hypothesis and, therefore, the related weights should be zeroed. After removing the $CACI'_{am}$ and $CMICI'_{am}$ terms, a

TABLE 12
Results of the Multilinear Regression Analysis of the CC'_{am} Metric for Total Effort Evaluation, Eff_{tot}

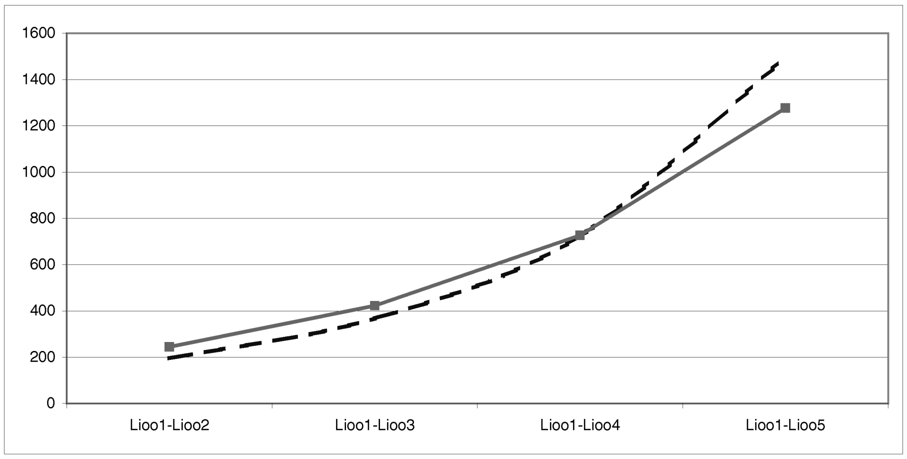
$CC'_{am} \approx Eff_{tot}$	w	$\mid t-value \mid$	$p-value$
$CACI'_b$	-0.642	1.028	0.306
$CACL'_b$	-0.117	2.939	0.004
$CMICI'_b$	0.327	1.323	0.188
$CMICL'_b$	2.199	12.891	0.000
$CACI'_a$	-0.473	0,790	0.431
$CACL'_a$	-0.110	2.816	0.006
$CMICI'_a$	0.361	1.488	0.139
$CMICL'_a$	2.346	16.048	0.000
R-squared	0.700		
F-stat (p-val)	36.147 (0.000)		
Correlation	0.881 with all coefficient		
	0.901 by removing $CACI'$ and $CMICI'$		
Std.Dev.	11.765 with all components		
	11.226 by removing $CACI'$ and $CMICI'$		

TABLE 13
Results of the Multilinear Regression Analysis of the NAM_{am} Metric for Total Effort Evaluation, Eff_{tot}

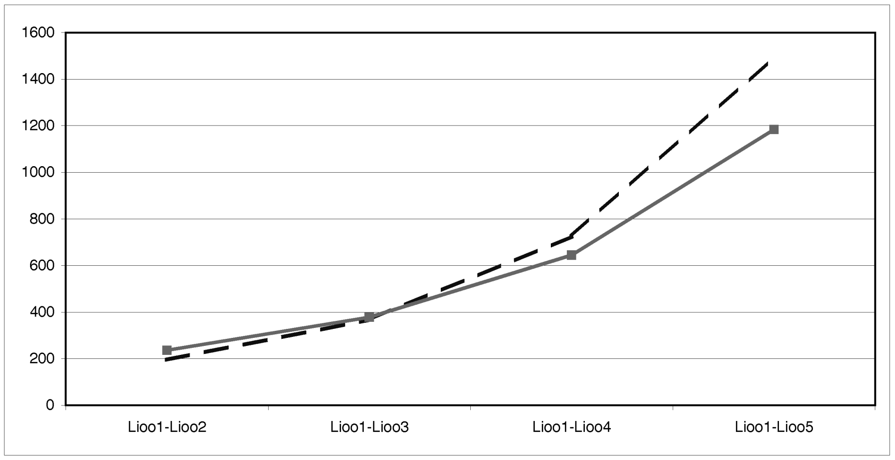
$NAM_{am} \approx Eff_{tot}$	w	$\mid t-value \mid$	$p-value$
NAI_b	2.188	1.979	0.050
NAL_b	2.331	2.300	0.023
NMI_b	-0.041	2.040	0.043
NML_b	1.378	5.443	0.000
NAI_a	0.510	0.503	0.616
NAL_a	3.334	3.485	0.001
NMI_a	0.064	0.319	0.751
NML_a	1.324	6.042	0.000
R-squared		0.558	
F-stat (p-val)		19.609 (0.000)	
Correlation		0.773	
Std.Dev.		10.230	

new evaluation of the correlation has produced a value of 0.90, with a standard deviation of 11.2. The more complete metric, CC_{am}, is better ranked with respect to the reduced CC'_{am}. This gives a very accurate effort prediction even in the early phases of the adaptive maintenance process. The $CMICL'$ term was the most significant for CC' estimation as resulted from the regression analysis. This can be justified by the fact that in the reengineering phase a great effort is devoted to the method interface, review, analysis, and understanding.

Metric NAM can be suitably used for effort prediction [11], while in this case, NAM has been used for predicting the adaptive maintenance effort. In Table 13, the results regarding multilinear regression analysis and correlation for total effort estimation of the adaptive maintenance for metric NAM_{am} are reported. A relatively high correlation has been obtained for NAM_{am}. The analysis performed on NAM_{am} highlights that NML is the most important factor among the metric components. This confirms the considerations about the relevance of the $CMICL_{am}$ term of CC_{am}, and the diffusion of the WMC version based on counting local methods. The terms of NMI have a specific behavior with respect to the other cases inspected since they present the same positive sign in both *after* and *before* terms (including the minus sign of (9)). On the other hand, this term is not very significant.



(a)



(b)

Fig. 8. Fitting trends for the intermediate versions from LIOO1 to LIOO5: (a) NAM_{am} and (b) CC'_{am}.

TABLE 14
Estimation Errors of Total Maintenance Effort (Eff_{tot}) by Using Predictive Metrics NAM_{am} and CC'_{am}

Metric(Reference Period)	$\lvert Error \rvert$	$\lvert MRE\% \rvert$
$NAM_{am}(LIOO1, LIOO2)$	49.1	20.0
$NAM_{am}(LIOO1, LIOO3)$	55.0	13.0
$NAM_{am}(LIOO1, LIOO4)$	2.22	0.3
$NAM_{am}(LIOO1, LIOO5)$	213.0	14.0
$CC'_{am}(LIOO1, LIOO2)$	40.9	17.3
$CC'_{am}(LIOO1, LIOO3)$	11.0	2.9
$CC'_{am}(LIOO1, LIOO4)$	80.4	12.5
$CC'_{am}(LIOO1, LIOO5)$	327.0	22.0

Several estimations depending on the application phase are reported.

Note that metric NAM_{am} is very cheap to calculate and the high correlation obtained means that it can produce a very interesting approximation of the adaptive maintenance effort when the method interfaces have not been defined. Metric NAM_{am} can be profitably used in the early phases of reengineering analysis. Once the method interfaces are defined, metric CC'_{am} can be used to obtain a better approximation of the maintenance effort. As soon as the same pieces of method code are written, CC_{am} and CC values can be adopted to evaluate the correct coefficients to be applied in future estimations. CC_{am} can be directly used since CC'_{am} can be regarded as a special case of CC_{am} when the functional parts are empty. Therefore, errors may be accepted in the predictive estimation of effort by using metric NAM_{am} since they are corrected when the estimation of NAM_{am} can be abandoned in favor of more precise metrics along the project life-cycle.

Metric $TJNAM_{am}$ can be considered equivalent to $NAML_{am}$ and presents a lower correlation with respect to the NAM_{am} and $Size2_{am}$ metrics, even if at system level it performs better than $Size2_{am}$, see Section 3.7. According to the above discussion, metric $Size2_{am}$ presents only one coefficient for the tuning process. The weight obtained by the multilinear regression analysis has the value of $w_{Size2_b} = 0.547$ with a great confidence ($p - value = 0.000$). A correlation of 0.671 has been obtained with a standard deviation of 8.58. This metric is penalized with respect to NAM for the lack of 1) inherited terms and of 2) weights associated with the metric terms.

NML_{am} produces quite good results considering its correlation (0.73) even though it is very simple to estimate.

In Fig. 8, the applications of metrics NAM_{am} and CC'_{am} to the intermediate versions of the system under adaptive maintenance is reported. As it is shown in the next sections these metrics are those that were better ranked among all metrics considered. The dashed line depicts the actual maintenance effort, while the continuous line represents the metric fitting: reevaluating weights for each intermediate version. In this way, the model has been validated not only between version 1 and 5 of LIOO, but also in all the intermediate versions. Version LIOO6 has not been taken into account since it is out of the maintenance as previously discussed.

In Table 14, the absolute error (in hours) and MRE percent, (100 (actual effort—predicted effort)/actual effort) in estimating the maintenance effort are reported.

TABLE 15
Results of the Multilinear Regression Analysis of CC'_{am} Metric for Code Manipulation Effort Evaluation, Eff_{cm}

$CC'_{am} \approx Eff_{cm}$	w	$\lvert t - value \rvert$	$p - value$
$CACI'_b$	-0.034	0.078	0.938
$CACL'_b$	-0.039	1.386	0.168
$CMICI'_b$	-0.002	0.009	0.993
$CMICL'_b$	1.645	13.720	0.000
$CACI'_a$	-0.078	0.185	0.853
$CACL'_a$	-0.037	1.352	0.006
$CMICI'_a$	0.049	0.285	0.776
$CMICL'_a$	1.552	15.112	0.000
R-squared	0.672		
F-stat (p-val)	31.828 (0.000)		
Correlation	0.811 with all coefficient 0.822 by considering only $CMICL'$		
Std.Dev	6.490 with all components 6.409 by considering only $CMICL'$		

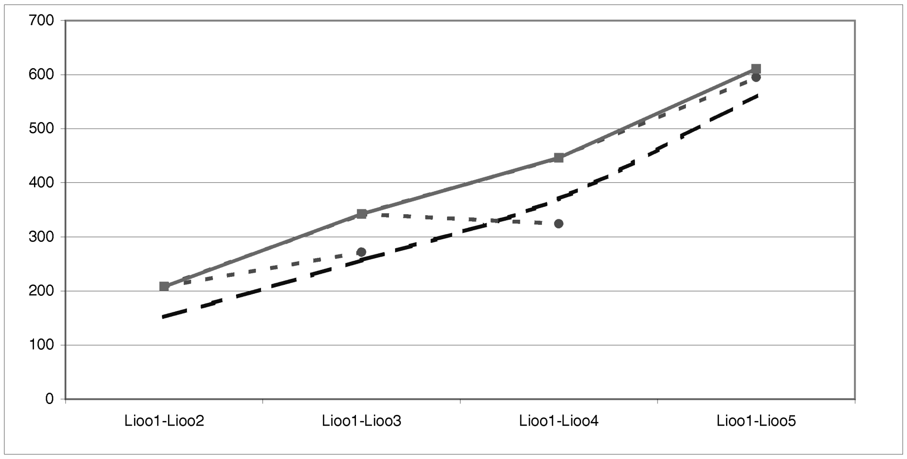
3.6 Predictive Estimation of Code Manipulation Effort of Maintenance

In Table 15, the multilinear regression analysis of CC'_{am} for predictive estimation of code manipulation effort is reported. The weights of $CMICI'$ present a different sign, but were nonsignificant for the model. In this case, the weights $w_{CMICL'}$ are the only significant weights of the metric; when all the other weights are set to zero, the correlation becomes slightly better. In this case, the correlation is greater than 0.8 and, thus, it can be suitably employed for prediction of code manipulation effort. It should be noted that the CC'_{am} metric is better ranked for total effort prediction. This confirms the suitability of the model and metric for effort prediction.

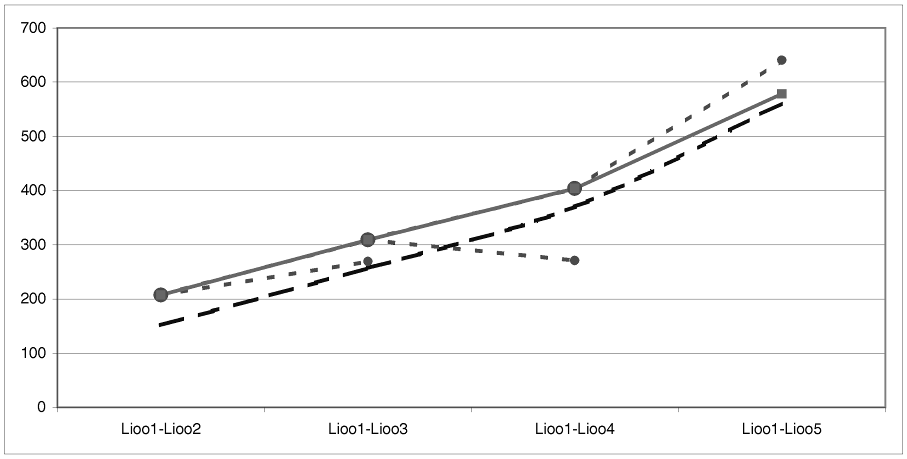
For metric NAM_{am}, the results are similar to those obtained for CC'_{am} (see Table 16). $TJNAM_{am}$ presents a lower correlation with respect to NAM_{am}. Metric $Size2_{am}$ confirms the good performance of metrics based on counting class members. By employing the same methodology described in Section 3.5 for the estimation of the

TABLE 16
Results of the Multilinear Regression Analysis of the NAM_{am} Metric for Code Manipulation Effort Evaluation, Eff_{cm}

$NAM_{am} \approx Eff_{cm}$	w	$\lvert t - value \rvert$	$p - value$
NAI_b	1.700	2.620	0.010
NAL_b	1.288	2.166	0.032
NMI_b	-0.378	3.050	0.003
NML_b	1.244	8.367	0.000
NAI_a	1.026	1.724	0.087
NAL_a	1.682	2.997	0.003
NMI_a	-0.199	1.674	0.097
NML_a	1.064	8.273	0.000
R-squared	0.663		
F-stat (p-val)	30.614 (0.000)		
Correlation	0.815		
Std.Dev.	6.256		



(a)



(b)

Fig. 9. Fitting trends and predictions for the intermediate versions from LIOO1 to LIOO5: (a) NAM_{am} and (b) CC'_{am}. The dashed line depicts the actual maintenance effort, while the continuous line represents the fitting: reevaluating weights for each intermediate version. The dotted lines depict the prediction activity from one phase to the next.

coefficient, a coefficient $w_{Size2} = 0.985$ and a correlation of 0.786 have been obtained with a standard deviation of 5.61. NML_{am} produces quite good results considering its correlation of 0.77.

In Fig. 9, the applications of metrics NAM_{am} and CC'_{am} for the estimation/prediction of code manipulation effort for the intermediate versions of the system under adaptive maintenance is reported. The dashed line depicts the actual maintenance effort, while the continuous line represents the fitting: reevaluating weights for each intermediate version.

In Table 17, the absolute error (in hours) and MRE percent in estimating the code manipulation effort of maintenance are reported. These values correspond to data depicted in Fig. 9 in continuous and dashed lines.

TABLE 17
Estimation Errors of Code Manipulation Maintenance Effort (Eff_{cm}) by Using the Predictive Metrics NAM_{am} and CC'_{am}

Metric(Reference Period)	$\|Error\|$	$\|MRE\%\|$
$NAM_{am}(LIOO1, LIOO2)$	56.5	27.2
$NAM_{am}(LIOO1, LIOO3)$	85.1	24.9
$NAM_{am}(LIOO1, LIOO4)$	75.9	17.0
$NAM_{am}(LIOO1, LIOO5)$	22.0	5.0
$CC'_{am}(LIOO1, LIOO2)$	55.5	26.8
$CC'_{am}(LIOO1, LIOO3)$	52.2	16.8
$CC'_{am}(LIOO1, LIOO4)$	33.6	8.3
$CC'_{am}(LIOO1, LIOO5)$	55.0	13.0

Several estimations depending on the application phase are reported.

TABLE 18
Errors in Predicting Code Manipulation Maintenance Effort (Eff_{cm}) by Using the Predictive Metrics Described in the Paper with the One-Step Prediction Metric

Metric(Reference Period,prediction)	$\|Error\|$	$\|MRE\%\|$
$NAM_{am}(LIOO1, LIOO2, LIOO3)$	14.6	5.4
$NAM_{am}(LIOO1, LIOO3, LIOO4)$	46.2	14.3
$NAM_{am}(LIOO1, LIOO4, LIOO5)$	33.7	5.7
$CC'_{am}(LIOO1, LIOO2, LIOO3)$	12.0	4.4
$CC'_{am}(LIOO1, LIOO3, LIOO4)$	99.5	36.7
$CC'_{am}(LIOO1, LIOO4, LIOO5)$	79.8	12.4

The metrics have been evaluated by using the weights estimated in the previous phase.

TABLE 19
Comparison Among Several Estimation Metrics for Maintenance Effort (Development and Total)

$M_{am}(LIOO1, LIOO5)$	Correlation	Std.Dev.	number of terms
$CC_{am} \approx Eff_{tot}$	0.899	11.50	12
$HSCC_{am} \approx Eff_{tot}$	0.635	10.22	6
$TJCC_{am} \approx Eff_{tot}$	0.626	10.09	4
$WMC \approx Eff_{tot}$	0.558	9.51	1
$CC_{am} \approx Eff_{cm}$	0.884	6.77	12
$HSCC_{am} \approx Eff_{cm}$	0.823	4.66	6
$TJCC_{am} \approx Eff_{cm}$	0.532	4.53	4
$WMC \approx Eff_{cm}$	0.442	3.98	1

TABLE 20
Comparison Among Several Predictive Metrics for Maintenance Effort (Development and Total)

$M_{am}(LIOO1, LIOO5)$	Correlation	Std.Dev.	number of terms
$CC'_{am} \approx Eff_{tot}$	0.901	11.22	8
$NAM_{am} \approx Eff_{tot}$	0.773	10.23	8
$TJNAM_{am} \approx Eff_{tot}$	0.765	11.21	4
$Size2_{am} \approx Eff_{tot}$	0.786	5.61	1
$NML_{am} \approx Eff_{tot}$	0.731	10.89	2
$CC'_{am} \approx Eff_{cm}$	0.822	6.41	8
$NAM_{am} \approx Eff_{cm}$	0.815	6.25	8
$TJNAM_{am} \approx Eff_{cm}$	0.794	6.33	4
$Size2_{am} \approx Eff_{cm}$	0.671	8.57	1
$NML_{am} \approx Eff_{cm}$	0.775	6.18	2

In order to verify the model predictability, a further inspection has been performed. In Fig. 9, the dotted lines represent the prediction activity from one phase to the next. In that case, the weights estimated by considering metric $M_{am}(ver.k, ver.k+1)$ between two consecutive versions have been used for predicting maintenance effort to reach version $k+2$, obtaining a one-step metric for prediction $M_{am}(ver.k, ver.k+1, ver.k+2)$. This approach poses the basis for the continuous control (prediction/estimation) of the maintenance process, where weights calculated in the previous phase are used to predict the next phase effort. In Table 18, the absolute errors and MRE percent with respect to the actual effort of maintenance for the predictions depicted in Fig. 9 are reported. Note that, even in this case, the prediction of adaptive maintenance is quite satisfactory. The maximum of the error has been recorded for the CC'_{am} metric.

3.7 Discussion and Comparison

In Tables 19 and 20, a comparison among the above-mentioned metrics when they are used for the estimation of the adaptive maintenance effort is reported. The comparison has been performed by using these metrics in the model of (9), obtaining in this way a set of new metrics: CC_{am}, $TJNAM_{am}$, CC'_{am}, NAM_{am}, NML_{am}, $Size2_{am}$, and WMC_{am}. The correlation and standard deviation values have been obtained via validation by considering the code of LIOO (after and before the adaptive maintenance) and the maintenance effort.

The results show that metrics $TJCC_{am}$ and $HSCC_{am}$ are less suitable than the complete model based on CC_{am} because they do not take into account the methods interface that has a great influence in the evaluation of adaptive maintenance effort. On the other hand, metric $Size2_{am}$ seems to have a relatively high correlation, considering that, in $Size2_{am}$, only local class members are considered. This fact is strengthened by the consideration that, in NAM_{am}, the most significant terms are related to the local part of the class.

In Table 21, the system estimated and/or predicted maintenance effort, S_X, is reported along with the absolute error (in hours) and MRE percent, in order to understand how a metric which performs better in terms of correlation can be suitably employed for the estimation of total system maintenance effort. In general, the metrics used underestimate the real effort and the error is more relevant to Total Effort that takes into account not only the code, but also all the maintenance related aspects.

As a general result, CC_{am} and NAM_{am} are the most suitable metrics for adaptive maintenance effort estimation and prediction. On the other hand, the other metrics considered also present quite satisfactory results. Note that the standard deviation of the estimating metrics is very close to that of effort, see Table 6.

In Table 22, results obtained by using the models of (10) and (11) have been reported for comparing them with those obtained for the same metrics with the more complete model of (9). The first part of the table reports the results obtained by using model $am0$ of (10) in which the measures on the code *after* the adaptive maintenance process have been used for estimating the corresponding effort. Note that metrics, such as $Size2$ and NAM, present very low correlation values. These are about 10 percent lower than those obtained by using the previously discussed model.

In the second part of Table 22, model $am1$ of (11) has been used for the estimation and prediction of the

TABLE 21
Comparison Among the Considered Metrics for System Estimation and Prediction of Effort (Code Manipulation and Total in Hours) for Adaptive Maintenance, Where $Eff_{tot} = 1,476$ Hours and $Eff_{cm} = 415$ Hours

X Metrics	S_X	$\|Error\|$	$\|MRE\%\|$
$CC_{am} \approx Eff_{tot}$	1258.4	218.0	14.8
$HSCC_{am} \approx Eff_{tot}$	792.2	684.2	46.3
$TJCC_{am} \approx Eff_{tot}$	622.6	853.7	57.8
$WMC \approx Eff_{tot}$	621.0	855.4	57.9
$CC'_{am} \approx Eff_{tot}$	1149.6	326.8	22.1
$NAM_{am} \approx Eff_{tot}$	1263.1	213.3	14.4
$TJNAM_{am} \approx Eff_{tot}$	944.0	532.2	36.1
$Size2_{am} \approx Eff_{tot}$	859.2	617.2	41.8
$NML_{am} \approx Eff_{tot}$	947.3	529.1	36.0
$CC_{am} \approx Eff_{cm}$	391.5	23.0	5.6
$HSCC_{am} \approx Eff_{cm}$	274.5	140.0	33.7
$TJCC_{am} \approx Eff_{cm}$	270.8	143.8	34.7
$WMC \approx Eff_{cm}$	243.4	171.2	41.3
$CC'_{am} \approx Eff_{cm}$	359.3	55.2	13.3
$NAM_{am} \approx Eff_{cm}$	392.4	22.2	5.3
$TJNAM_{am} \approx Eff_{cm}$	304.2	110.1	27.1
$Size2_{am} \approx Eff_{cm}$	297.2	117.3	28.3
$NML_{am} \approx Eff_{cm}$	306.1	109.0	26.2

maintenance effort. In this model, *after* and *before* terms present the same weights.

Note that both these models are less satisfactory than that obtained by using metric model M_{am} of (9). Similar results are confirmed by all the other metrics considered.

The correlation values obtained between maintenance effort with CC_{am0} and NAM_{am0} and $Size2_{am0}$ are less satisfactory than those obtained for other metrics. The results demonstrate that classical metrics for estimation and prediction of development effort are unsuitable for the estimation of the adaptive maintenance effort with respect to the newly identified metrics and model. They can be suitably employed for maintenance effort estimation by using the model of (9).

4 Conclusions

A general model for adaptive maintenance effort evaluation and prediction has been proposed. The application of the model to metrics and their validation with respect to real data collected during the ESPRIT project MOODS have allowed us 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/size metrics for the estimation of development effort presented in the literature. Statistical validations and the estimated weights have been reported.

The proposed model has been compared with simpler models considering several different metrics. The results obtained have shown that the proposed model is better ranked with respect to the simpler models. The difference results from a 10 percent to 20 percent of increment in correlation. The model can be also used for predicting maintenance effort depending on the metric used.

The main lessons learned from this analysis can be summarized in the following points:

1. During adaptive maintenance, strong attention should be paid to method definition and in particular to the method interfaces.
2. Local attributes are not so relevant during the adaptive maintenance (typically few attributes are added), while inherited attributes are relevant since they hide internal states for subclasses instances.
3. CC'_{am} was the most suitable metric for predicting the effort for the adaptive maintenance (the counting of the number of methods or the estimation of $Size2$ metrics can be good compromises for an early system assessment).
4. The complexity of the interfaces for locally defined methods and their number are the most important factors for estimating the adaptive maintenance effort, their complexity is a cost for comprehending the system and thus for its manipulation.
5. The analysis of the evolution of metric weights can help to understand the process performed on a system.

TABLE 22
Results Obtained by Using Other Two Classical Metric Models for Estimation of Maintenance Effort

	$Eff_{am} \approx M_{am0}$		
Metric M used	Correlation	Std. Dev.	number of terms
$CC_{am0} \approx Eff_{tot}$	0.632	9.531	6
$WMC_{am0} \approx Eff_{tot}$	0.547	9.380	1
$NML_{am0} \approx Eff_{tot}$	0.346	3.074	1
$NAM_{am0} \approx Eff_{cm}$	0.496	7.803	4
$Size2_{am0} \approx Eff_{cm}$	0.519	29.392	1
$NML_{am0} \approx Eff_{cm}$	0.505	8.833	1

	$Eff_{am} \approx M_{am1}$		
Metric M used	Correlation	Std. Dev.	number of terms
$CC_{am1} \approx Eff_{tot}$	0.861	12.249	6
$WMC_{am1} \approx Eff_{tot}$	0.136	3.278	1
$NML_{am1} \approx Eff_{tot}$	0.745	6.202	1
$NAM_{am1} \approx Eff_{cm}$	0.757	11.312	4
$Size2_{am1} \approx Eff_{cm}$	0.769	11.290	1
$NML_{am1} \approx Eff_{cm}$	0.775	11.026	1

TABLE 23
Glossary of the Metrics Mentioned in this Paper

metric	description
$CACI$	Class Attribute Complexity/size Inherited
$CACL$	Class Attribute Complexity/size Local
CC	Class Complexity/size
CC'	Class Complexity/Size predictive form
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
Eff_{add}	Effort related to the addition of code during the adaptive maintenance
Eff_{am}	Total Effort spent for adaptive maintenance
Eff_{chang}	Effort related to the change of code during the adaptive maintenance
Eff_{cm}	Effort spent during the process of Adaptive Maintenance for code modification
Eff_{del}	Effort related to the deletion of code during the adaptive maintenance
Eff_{tot}	Total Effort related to the Adaptive maintenance
Eff_{und}	Effort related to the system understanding before the adaptive maintenance
Ha	Halstead metric [?]
$HSCC$	Class Complexity like [?]
ICI	Internal Class Implementation
LOC	number of Lines Of Code
M_{am}	Model am for Adaptive Maintenance Metrics
M_{am0}	Model $am0$ for Adaptive Maintenance Metrics
M_{am1}	Model $am1$ for Adaptive Maintenance Metrics
M_a	Generic Metric estimated after the process of adaptive maintenance
M_b	Generic Metric estimated before the process of adaptive maintenance
M_{del}	Generic Metric estimated on the code deleted during the adaptive maintenance
M_r	Generic Metric estimated on reused code
MCC	Mean value of Class Complexity
MNA	Mean Number of Class Attributes
MNM	Mean Number of Class Methods
NA	Number of Attributes of a class (inherited and locally defined)
NAI	Number of Attributes Inherited of a class
NAL	Number of Attributes Locally defined 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
NM	Number of Methods of a class (inherited and locally defined)
NMI	Number of Methods Inherited of a class
NML	Number of Methods Local of a class
NRC	Number of Root Classes in the system class tree
S_X	System complexity based on X metric
$Size2$	Number of class attributes and methods [?]
T	generic additive term of a metric
$TJCC$	Class Complexity like [?]
$TJNAM$	Class Size like [?]
$TLOC$	Total (System) Number of LOC
TNM	Total (System) Number of Methods
Vg'	McCabe ciclomatic Complexity
WMC	Weighted Methods for Class [?]

6. Metrics based on counting members are less precise than functional-based metrics.
7. Total effort of maintenance can be better estimated by using functional-based metrics since metrics based on counting class members neglect several aspects that produce a strong impact on documentation and test costs.
8. Simple development metrics used in model $am0$ are unsuitable to estimate maintenance effort; the best model was am, the proposed one.

APPENDIX

The Appendix presents a glossary of the metrics mentioned in this paper (Table 23).

ACKNOWLEDGMENTS

The authors would like to thank Professor G. Bucci for his suggestions and the members of the MOODS team for the collected data and the reviewers for their contribution in making the paper stronger and clearer. They extend a particular thanks to the reviewers that provided useful suggestions to improve the paper and to the people who have worked in the past for the implementation and testing of the several components of our code analyzer, TAC++. The code analyzer for the estimation of the metrics mentioned in this paper can be obtained by sending a request to the authors. This work was partially supported by Ministero dell'Universita' e della Ricerca Scientifica e Tecnologica (MURST Ministry of University and Scientific Research).

REFERENCES

[1] N.F. Schneidewind, "The State of Software Maintenance," *IEEE Trans. Software Eng.*, vol. 13, no. 3, pp. 303–310, Mar. 1987.

[2] D. Kafura and G.R. Reddy, "The Use of Software Complexity Metrics in Software Maintenance," *IEEE Trans. Software Eng.*, vol. 13, no. 3, pp. 335–343, Mar. 1987.

[3] C.F. Kemerer and S. Slaughter, "An Empirical Approach to Studying Software Evolution," *IEEE Trans. Software Eng.*, vol. 25, no. 4, pp. 493–509, Apr. 1999.

[4] D. Coleman, D. Ash, B. Lowther, and P. Oman, "Using Metrics to Evaluate Software System Maintainability," *Computer*, pp. 44–49, Aug. 1994.

[5] M. Jorgensen, "Experience with the Accuracy of Software Maintenance Task Effort Prediction Models," *IEEE Trans. Software Eng.*, vol. 21, no. 8, pp. 674–681, Aug. 1995.

[6] R.D. Banker, S.M. Dtar, C.F. Kemerer, and D. Zweig, "Software Complexity and Maintenance Costs," *Comm. ACM*, vol. 36, no. 11, pp. 81–94, Nov. 1993.

[7] B.W. Boehm, *Software Engineering Economics*, Prentice Hall, 1982.

[8] B. Kitchenham, "Empirical Studies of Assumptions that Underlie Software Cost-Estimation Models," *Information and Software Technology*, vol. 34, pp. 211–218, Apr. 1992.

[9] T. Mukhopadhyay and S. Kekre, "Software Effort Models for Early Estimation of Process Control Applications," *IEEE Trans. Software Eng.*, vol. 18, no. 10, pp. 915–923, Oct. 1992.

[10] G.K. Gill and C.F. Kemerer, "Cyclomatic Complexity Density and Software Maintenance Productivity," *IEEE Trans. Software Eng.*, vol. 25, no. 12, pp. 1284–1288, Dec. 1999.

[11] P. Nesi and T. Querci, "Effort Estimation and Prediction of Object-Oriented Systems," *The J. Systems and Software*, vol. 42, pp. 89–102, 1998.

[12] V.R. Basili, L. Briand, and W. L. Melo, "A Validation of Object Oriented Design Metrics as Quality Indicators," *IEEE Trans. Software Eng.*, pp. 751–761, Oct. 1996.

[13] C.F. Kemerer, "An Empirical Validation of Software Cost Estimation Models," *Comm. ACM*, vol. 30, no. 5, pp. 416–429, May 1987.

[14] W. Li and S. Henry, "Object-Oriented Metrics that Predict Maintainability," *The J. Systems Software*, vol. 23, pp. 111–122, 1993.

[15] W. Li and S. Henry, "Maintenance Metrics for the Object Oriented Paradigm," *Proc. First IEEE Int'l Software Metrics Symp.*, pp. 52–60, May 1993.

[16] B. Henderson-Sellers, *Object Oriented Metrics.* N.J.: Prentice Hall, 1996.

[17] P. Nesi and M. Campanai, "Metric Framework for Object-Oriented Real-Time Systems Specification Languages," *The J. Systems and Software*, vol. 34, pp. 43–65, 1996.

[18] L.C. Briand, J.W. Daly, and J. K. Wust, "A Unified Framework for Coupling Measurement in Object Oriented Systems," *IEEE Trans. Software Eng.*, vol. 25, no. 1, pp. 91–120, Jan./Feb. 1999.

[19] L.C. Briand, J. Wust, and H. Lounis, "Using Coupling Measurements for Impact Analysis in Object Oriented Systems," *Proc. IEEE Int'l Conf. Software Maintenance*, Sept. 1999.

[20] L.C. Briand, J. Wust, J.W. Daly, and D.V. Porter, "Exploring the Relationships between Design Measures and Software Quality in Object Oriented Systems," *J. Systems and Software*, 1998.

[21] P. Nesi, "Managing Object Oriented Projects Better," *IEEE Software*, pp. 50–60, July/Aug. 1998.

[22] S.N. Cant, B. Henderson-Sellers, and D.R. Jeffery, "Application of Cognitive Complexity Metrics to Object-Oriented Programs," *J. Object Oriented Programming (JOOP)*, pp. 52–63, July/Aug. 1994.

[23] D. Thomas and I. Jacobson, "Managing Object-Oriented Software Engineering," *Tutorial Note, TOOLS '89, Int'l Conf. Technology of Object-Oriented Languages and Systems*, p. 52, Nov. 1989.

[24] B. Henderson-Sellers, "Some Metrics for Object-Oriented Software Engineering," *Proc. Int'l Conf. Technology of Object-Oriented Languages and Systems, TOOLS 6*, pp. 131–139, 1991.

[25] S.R. Chidamber and C.F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Trans. Software Eng.*, vol. 20, no. 6, pp. 476–493, June 1994.

[26] B. Henderson-Sellers, "Identifying Internal and External Characteristics of Classes Likely to be Useful as Structural Complexity Metrics," *Proc. Int'l Conf. Object Oriented Information Systems, OOIS '94*, D. Patel, Y. Sun, and S. Patel, eds., pp. 227–230, Dec. 1994.

[27] L.A. Laranjeira, "Software Size Estimation of Object-Oriented Systems," *IEEE Trans. Software Eng.*, vol. 16, no. 5, pp. 510–522, May 1990.

[28] M. Lorenz and J. Kidd, *Object-Oriented Software Metrics, A Practical Guide.* N.J.: PTR Prentice Hall, 1994.

[29] F. Fioravanti, P. Nesi, and S. Perlini, "A Tool for Process and Product Assessment of C++ Applications," *Proc. Second Euromicro Conf. Software Maintenance and Reeng.*, pp. 89–95, Mar. 1998.

[30] G. Bucci, F. Fioravanti, P. Nesi, and S. Perlini, "Metrics and Tool for System Assessment," *Proc. IEEE Int'l Conf. Complex Computer Systems*, pp. 36–46, Aug. 1998.

[31] F. Fioravanti, P. Nesi, and S. Perlini, "Assessment of System Evolution through Characterization," *Proc. IEEE Int'l Conf. Software Eng.*, pp. 456–459, Apr. 1998.

[32] F. Fioravanti and P. Nesi, "A Method and Tool for Assessing Object-Oriented Projects and Metrics Managemen," *The J. Systems and Software*, 2001.

[33] L. Briand, J. Wurst, S. Ikonomovski, and H. Lounis, "A Comprehensive Investigation of Quality Factors in Object Oriented Designs: An Industrial Case Study," Technical Report ISERN-98-29, IESE-47988e, IESE, Germany, 1998.

[34] F. BritoeAbreu, M. Goulao, and R. Esteves, "Toward the Design Quality Evaluation of Object Oriented Software Systems," *Proc. Fifth Int'l Conf. Software Quality*, Oct. 1995.

[35] O. Signore and M. Loffredo, "Some Issues on Object-Oriented Re-Engineering," *Proc. ERCIM Workshop Methods and Tools for Software Reuse*, 1992.

[36] M.A. Chaumun, H. Kabaili, R.K. Keller, and F. Lustman, "A Change Impact Model for Changeability Assessment in Object Oriented Software Systems," *Proc. Second Euromicro Conf. Software Maintenance and Reeng.*, Mar. 1999.

[37] P. Bengtsson and J. Bosh, "Architecture Level Prediction of Software Maintenance," *Proc. Third Euromicro Conf. Software Maintenance and Reeng.*, Mar. 1999.

[38] P.J. Rousseeuw and A.M. Leroy, *Robust Regression and Outlier Detection.* N.Y.: John Wiley & Sons, 1987.

[39] T.J. McCabe, "A Complexity Measure," *IEEE Trans. Software Eng.*, vol. 2, no. 4, pp. 308–320, 1976.

[40] B. Henderson-Sellers and J.M. Edwards, "The Object Oriented Systems Life Cycle," *Comm. ACM*, vol. 33, no. 9, pp. 143–159, Sept. 1990.

[41] H.M. Halstead, *Elements of Software Science.* N. Holland: Elsevier, 1977.

[42] F. Fioravanti and P. Nesi, "Complexity/Size Metrics for Object-Oriented Systems," Technical Report, Univ. of Florence, TR 17/99, Italy, 1999.

[43] S.R. Chidamber, D.P. Darcy, and C.F. Kemerer, "Managerial Use of Metrics for Object Oriented Software: An Exploration Analysis," *IEEE Trans. Software Eng.*, vol. 24, no. 8, pp. 629–639, Aug. 1998.

[44] G. Booch, *Object-Oriented Design with Applications.* Calif.: The Benjamin/Cummings Publishing Company, 1994.

Fabrizio Fioravanti received the Laurea Degree in electronic engineering and the PhD degree in software engineering and telecommunications from the University of Florence, Italy. He is an assigned professor of computer architecture at the same University. He has been local chair and a member of the program committee of international conferences. His current research interests include software engineering, object-oriented technologies, and software metrics for quality estimation of object-oriented systems. He is a member of the IEEE and the IEEE Computer Society.

Paolo Nesi received the DrEng degree in electronic engineering from the University of Florence, Italy. He received the PhD degree from the University of Padoa, Italy, in 1992. In 1991, he was a visitor at the IBM Almaden Research Center, California. Since 1992, he has been with the Dipartimento di Sistemi e Informatica, where he is a professor for the University of Florence, Italy. He is active on several research topics: formal methods, object-oriented technology, real-time systems, system assessment, physical models, and parallel architectures. He has been general chair, program chair, or cochair of some international conferences. He is a member of the program committee of several international conferences, among them: IEEE ICECCS, IEEE ICSM, IEEE METRICS, CSMR, etc. He is the general chair of the IEEE International Conference on Software Maintenance, 2001. He has been the guest editor of special issues of international journals and an editorial board member of the *Journal of Real-Time Imaging* and of a book series of CRC. He is the author of more than 120 technical papers and books. He has been the coordinator of several ESPRIT projects and holds the scientific responsibility at the CESVIT (High-Tech Agency for technology transfer) for object-oriented technologies and HPCN. He is a member of the IEEE, the IEEE Computer Society, the ACM, ICMA, AIIA, and TABOO.