Figure 3: TAC++ class tree browser

To perform this process, the user can decide to assess or not class libraries (if any). In any case, the definitions (C++ headers files) of the libraries used must be available. If the code of class library is available, it can be considered into the assessment process or not.

The class browser provides an integrated editor and uses the multi-linked structure for navigating among system classes and their members. As can be observed in Fig.3, the browser shows the list of classes with a synthetic description of their relationship: the class hierarchy, the list of methods, etc. By selecting a class and its method, the corresponding method code is directly available in separate editing windows. Each change in the class definition and/or implementation may produce corresponding changes in the data contained in the multi-linked structure.

The above-described process is capable of producing Low-Level Metrics (LLM). These are direct metrics – such as LOC, Mc, Ha, NA, NAL, NAI, NM, NMI, NML, MIC, CBO, etc. LLMs Evaluator saves on a file the multi-linked structure. This comprises also the description of each method in terms of its parameters and definition of local variables. The same approach is followed for global definitions, variable and subprograms that can be present in C++ systems even if their adoption is not properly coherent with the OOP.

The results produced by the LLMs Estimator can be collected on the basis of the life-cycle phase in which they are estimated. Please note that, most of the above mentioned direct metrics can produce draft results even when only the class definition is available, such as in the early analysis. This can be very useful for predicting final values of these metrics and thus for predicting the final product characteristics.

4.2 Defining and Collecting High-Level Metrics

High-Level Metrics (HLMs) can be defined by the user on the basis of LLMs. To this end, a specific interactive tool allows the definition of new metrics by means of a visual editor (see Fig.4). HLMs can be defined according to the following structure comprised of one or more additive terms:

$$NewMetric = W_{M1}M1\frac{W_{U1}\sum_x U1}{W_{D1}\sum_x D1} + ... + W_{Mn}Mn\frac{W_{Un}\sum_x Un}{W_{Dn}\sum_x Dn}$$

where x is the context in which the sum is performed – for instance (i) on all system classes, (ii) on all class methods, (iii) on all class attributes, etc. Each sum on x can be set to operate on a single value,

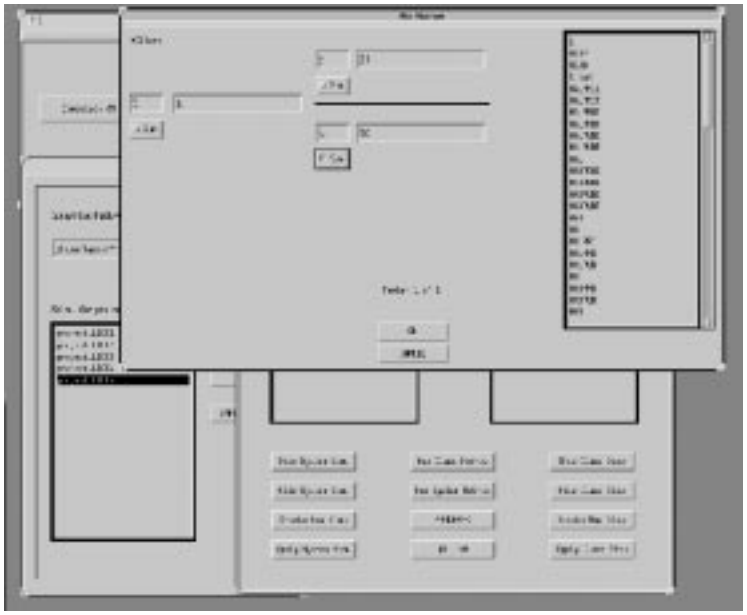

Figure 4: High Level Metric Editor

thus transforming the terms in a single term.

Weights can be imposed on the basis of company experience and goals by using a set of reference projects with the aim of the related database of weights. Entries of the database weights are evaluated during the various phases of system development, or set to fixed values. Please note that for each metric, different values of weights can be saved according to the life-cycle phase in which they are collected and significant. This is because different values for the weights have to be usually registered and used in different phases of the software life-cycle.

The HLMs Evaluator obtains the values of indirect metrics on the basis of the current definitions of HLMs and by using LLMs values. The values of HLMs are stored together with the values of their weights and phase in which they have been assessed.

4.3 Visualizing Results

In order to provide a fast and understandable view of the project status, the values obtained for LLMs and HLMs can be collected for visualizing them in a set of specific views. The views can be defined for satisfying the needs of developers as well as those of subsystem and project managers. These are typically directly defined in the internal development manual of the company together with many other rules that must be followed for developing the system (i.e., manual of development of the company). Different views will be defined for addressing the specific needs of developer, subsystem and project managers. Views can be histograms or profiles.

4.3.1 Profiles

A profile is a consumptive view, which is capable of showing the values of several metrics with respect their specific mean, maximum, minimum and acceptable values. The minimum/maximum value(s) can be considered the lowest/greatest value(s) under/over of which a correction should be needed.

These views are shown by using normalized graphs: Kiviat, bar, pie, etc. by means of the Viewer (see Fig.5). In order to make faster the correction for solving undesirable lowest and greatest values, a brief comment describing what could be done is associated to each HLM directly in the HLMs database. This places the basis for the continuous improvement of system quality and conformity with the required product and process profile.

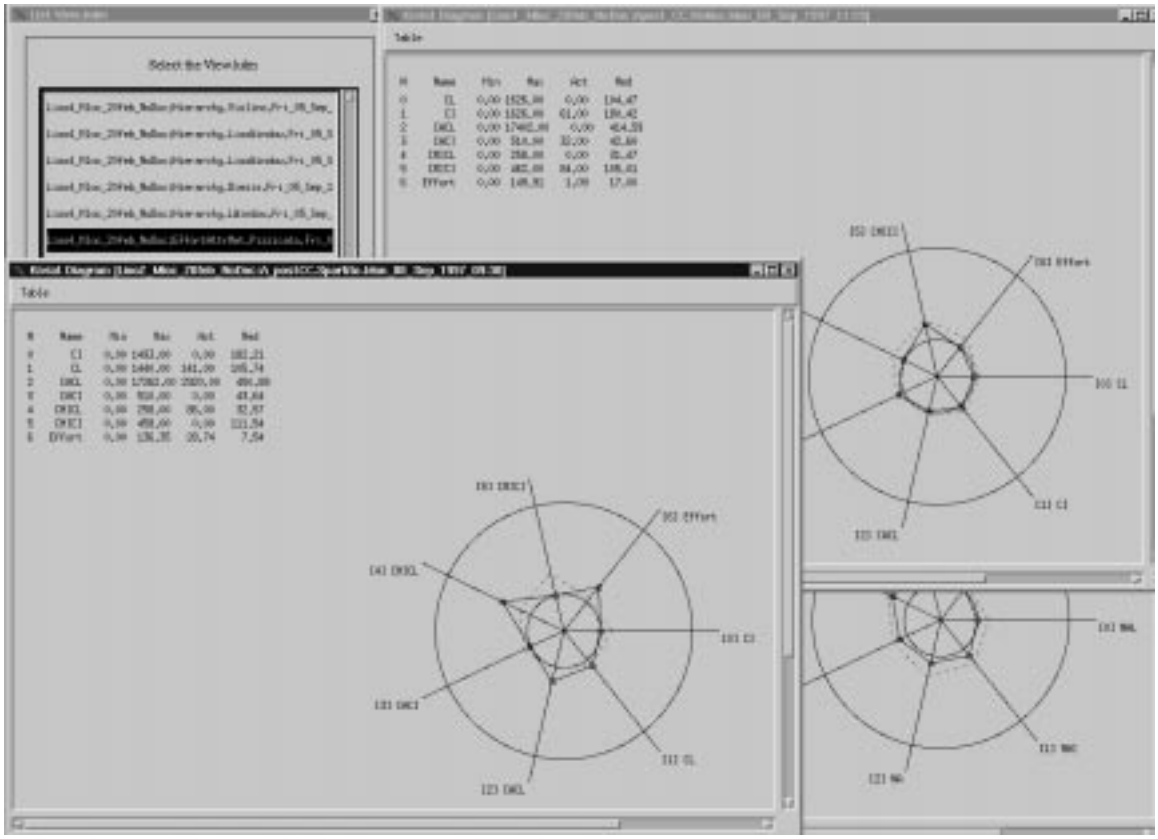

Figure 5: TAC++ Kiviat Diagram for a selected view.

For example, in order to monitoring class quality a view reporting values of metrics: NA, NM, CC_m, CM_m/NM, $CCGI$, $NSUB$, DIT, $NSUP$, etc., can be defined. Other examples of typical views are: (i) a view reporting the effort prediction of a class: $Size2$, NAM, WMC, CC', etc.; (ii) a view presenting effort estimation of a class: CC'_m, CC_m, WMC, Mc, etc.; (iii) views reporting effort prediction or estimation at system level: SC, T_{LOC}; (iv) views reporting conformity to OOP at system level: NRC, NRC/NCL, SC_m/NCL, etc.; (v) views reporting class metrics related to reusability and maintainability: NOC, $NSUP$, $NSUB$, DIT, etc.

In addition, in this case, the definitions of views are organized according to the life-cycle phase for which they have been defined. In this way, reference values, and actions for solving problems can be imposed phase by phase, reporting specific values and comments.

4.3.2 Histograms

By using TAC++ it is also possible to evaluate the statistic view of each metrics for the system under assessment. For example: (i) the number of classes for the complexity of classes, (ii) the number of methods for the complexity of methods, (iii) the number of classes for their $CCGI$, (iv) the number of classes for their $NSUP$, (v) the number of classes for their $NSUB$, etc. (see Fig.6). These histograms are useful for identifying which classes are outside of the company established or recommended bounds. In fact, typical trends of histograms can be assumed as reference patterns by the company. In some cases the histogram trends is also independent of the languages [21], [22].

A collection of histograms among the various phases of development life cycle could aid the system manager to take into account the modification, from the point of view of quality profile, of each class in the system.

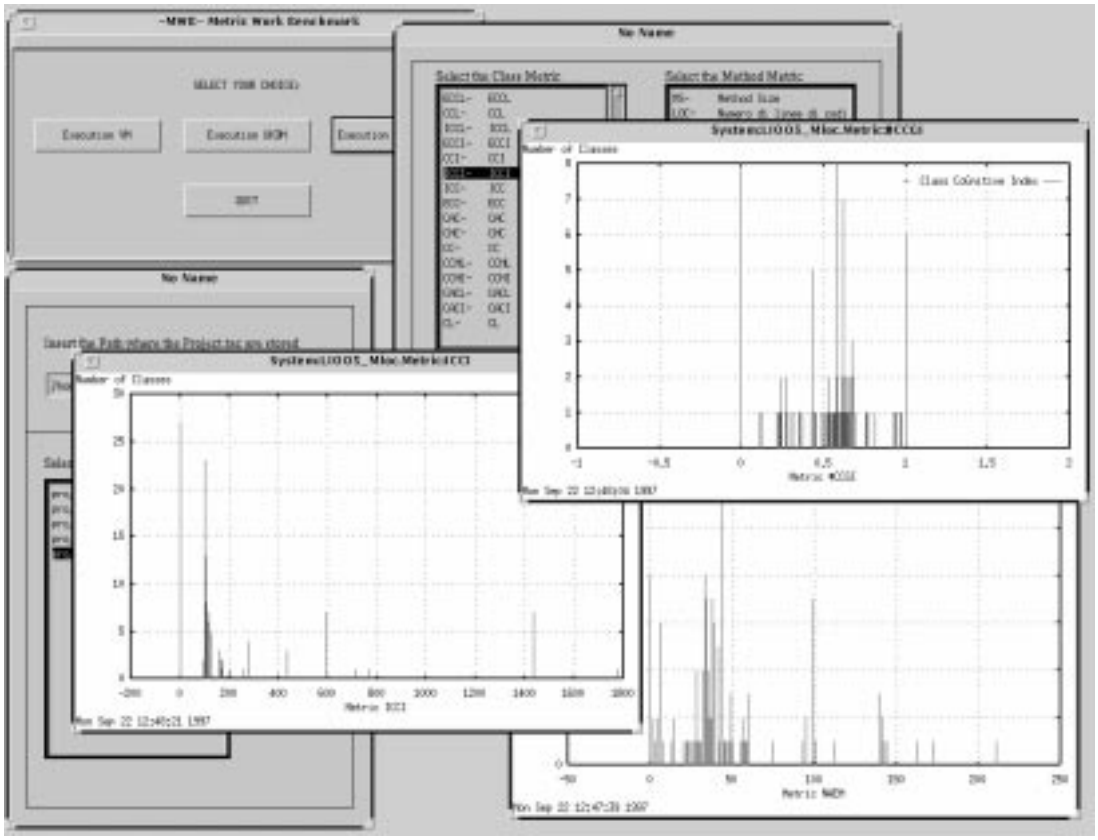

Figure 6: TAC++ generated histogram.

4.4 Validating and Tuning Metrics

Despite to the high number of metrics, in the literature, there exists only few papers reporting accurate validations of metrics [14], [9]. The adoption of a validation technique is not only a work for researchers. It mainly consists in identifying metric parameters (weights) on the basis of the knowledge of actual data – e.g., on the basis of the real effort of development as well as of maintenance or the number of defects encountered in developing a class, etc., depending on the goals of the metric under validation.

The validation process can be used for (i) verifying which terms of each identified metric is relevant for its estimation (this can be used for reducing the estimation effort and, in some cases, for increasing correlation and reliability), (ii) evaluating the confidence of the measures obtained, (iii) tuning metric models according to different context and profiles (weights and scale identification), (iv) identifying metric parameters along the development life-cycle for evaluating the development progress with respect to reference trend, and finally (v) for evaluating the goodness of metrics in representing the selected feature (i.e., selecting metrics).

For these reasons, the validation and tuning processes must be performed at different levels, by the developers, by the subsystem and project managers. In particular, operation (iv) must be performed by the developers, while operations (ii), (iii), (v) by the subsystem and project managers and (i) by specific people devoted to the definition of methods and instruments for controlling the process of development.

Usually, the validation can be performed by using mathematical and statistical techniques such as multilinear regression tools [26] or logistic regression [27]. In both cases, real data reporting direct measures of the features that should be evaluated by metrics are needed – e.g., real effort, number of defects identified, results of objective test about usability, etc. This information is collected by using a Data Collector, which in our tool is developed in Java to be portable in a wide number of platforms.

Weight values depend on the application context and, thus, it is possible to obtain more precise

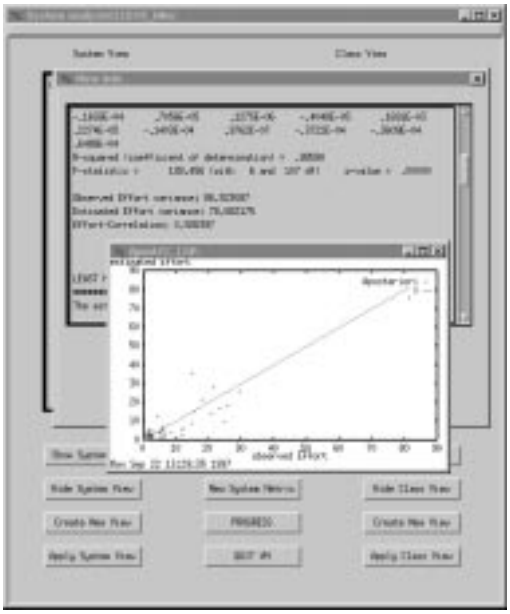

Figure 7: TAC++ Statistical Tool

results by estimating the weights depending on the type of the system under development and for each specific phase of the development life-cycle. This can be simply done by using a little number of reference projects into the selected area and estimating weights with the previously applied method. The reference projects must comply with OOP and quality profiles requested for the project area.

The Statistic Analyzer is capable of estimating all the metric weights used in a metrics if the corresponding real value of the metric is available – e.g., if CC_{Mc} is considered, and the real effort for each class is available, the statistical tool can estimate the optimal weights by means of multilinear robust regression. The values of the weights can be used for assessing other systems and as a starting value for defining reference values of the company.

Usually a metric may present a high number of components but not all the terms have the same importance. By using the Statistic tool, it is possible to verify not only the correlation of the whole metric with respect to the real data, but also the correlation of each term of the metric with respect to the collected effort, maintenance or other real data. Moreover, the statistic tool is capable of identifying the metrics components that are more relevant for the estimation of the targeted features. This is performed by considering p-values and t-values of multilinear regression analysis [26]. In this way, a process of refinement can be performed in order to identify whose terms of the metric are more significant than the others for obtaining the indirect measure; for example by observing the influence of each metric component in estimating or predicting effort, maintenance, etc.

Please note that, the most important metrics of TAC++ have been analyzed and validated by using the above technique, during the development of several C++ projects. This validation process has been performed in order to discover whose metrics must be evaluated and which components are relevant [9], and when they give good results for controlling the system main characteristics.

The engine of Statistic Analyzer is mainly based on multilinear regression techniques [26] (Progress). Since Progress tool presents a textual interface, a graphical user interface has been added to make it more user friendly. Moreover, the result of multilinear regressions can be easily interpreted since they can be graphically visualized as dot diagrams (see Fig.7).

A Posteriori	Corr.	Variance
$WMC = CL_{Mc}$ [20]	0.90	245
CM_{LOC} [22]	0.91	186
CM_{Ha} [16]	0.82	423
CC_{Mc} [9]	0.93	149
CC_{Ha} [9]	0.94	216
CC_{LOC} [9]	0.94	145
$TJCC$ [19]	0.93	157
$HSCC$ [10]	0.93	146

Predictive	Corr.	Variance
CC'_{LOC} [9]	0.88	770
NAM [18]	0.73	1100
$Size2$ [6]	0.72	1700

Table 2: Comparison between metrics defined for evaluation and prediction of class effort.

4.5 Experiments on Effort Evaluation and Prediction

In Tab.2, the most diffuse metrics for effort estimation at level of class are compared against the well known pure functional metrics. The comparison is made on the basis of correlation and variance values obtained by the TAC++ Statistic Analyzer (confidence values and values of weights have been omitted) (see [9] for details of the validation process).

Lower values of variance correspond to a less spread distribution – i.e., a lower probability to get the wrong effort estimation or prediction by using the selected metric. The analysis reported shows that traditional functional metrics can be profitably employed for evaluating object-oriented systems if they are used as a basis for more complete metrics (as in [22], [28]). The metrics proposed in [9] present both a higher value of correlation and a lower value of variance. The differences among the values of correlation are not so strong and, thus, the values of variance are important since give an idea of the estimation confidence. These values have been obtained by assessing several C++ applications with a high degree of conformance to the OOP [9].

During the system assessment, particular attention must be paid to applications that adopt an object-oriented library of classes for managing GUI with windows (e.g., Motif plus CommonView, MS-Windows plus MSVC++, and so on). Usually these projects have approximately 30% of code devoted to user interface management and, therefore, this property must be taken into account during the weight and metric bounds tuning.

In order to estimate the trend of selected metrics and weights for subsequent evaluations, it is very important to record values during the early phases of the system life-cycle. Once the weights are identified, the adoption of predictive metrics is very useful, see Tab.2 in which the correlation values obtained for these metrics are very encouraging, because small errors can be accepted in the early phase of software development cycle. By using TAC++ a collection of trends of development of some projects have been recorded by the research workgroup of the authors in order to establish bounds for metrics and specific values for the weight depending on the application field. This experience is going to revealing every day more effective since other projects under assessment are following the values predicted in the early phases. Moreover, the tool has demonstrated it usefulness for detecting degenerative conditions in the class hierarchy, thus controlling the maintainability, reusability, etc.

5 Discussion and Conclusions

The adoption of the OOP has produced a great demand of specific metrics. Several direct and indirect metrics for the evaluation of effort, maintenance and reusability costs, have been defined. Therefore, in order to manage this large number of metrics, an integrated tool for defining, showing and validating

them is mandatory.

The tool presented in this paper offers to developers, subsystem and project managers a highly configurable environment to control all the aspects of a C++ projects since the early phase of system development. TAC++ offers many features for aiding project development and maintenance: (i) direct manipulation of code, (ii) low-level metric evaluation, (iii) High level metric definition and evaluation, (iv) graphical representation of metric profiles and histogram, (v) metrics validation and tuning by the means of a multilinear regression engine, and (vi) real data collector – i.e. real effort in person-hours, number of errors, number of fault, etc.

TAC++ has been profitably used for controlling the development and maintenance of several projects elaborated by the research group of the authors in order to validate its use on the application field. The results obtained by the above mentioned tests have permitted to the research group to establish bounds, reference profiles and histograms for a wide typology of applications, that will be used for develop present and future C++ projects.

Acknowledgments

The authors would like to thank all the members of TAC++ team and the many other people involved in its development in the past. A special thank to B. Pages for an early version of the class browser (i.e., Xcoral), and A. Borri of CESVIT, for his help in designing the first version of TAC++.

References

[1] B. Henderson-Sellers and J. M. Edwards, "The object oriented systems life cycle," *Communications of the ACM*, vol. 33, pp. 143–159, Sept. 1990.

[2] B. W. Boehm, "A spiral model of software development and enhancement," *ACM SIGSOFT Software Engineering Notes*, vol. 11, no. 4, pp. 14–24, 1986.

[3] P. Nesi, *Objective Software Quality, Proc. of Objective Quality 1995, 2nd Syposium on Software Quality Techniques and Acquisition Criteria.* Berlin: Lecture Notes in Computer Science, N.926, Springer Verlag, 1995.

[4] B. Henderson-Sellers, D. Tegarden, and D. Monarchi, "Metrics and project management support for an object-oriented software development," in *Tutorial Notes TM2, TOOLS Europe'94, International Conference on Technology of Object-Oriented Languages and Systems*, (Versailles, France), 7-10 March 1994.

[5] L. A. Laranjeira, "Software size estimation of object-oriented systems," *IEEE Transactions on Software Engineering*, vol. 16, no. 5, pp. 510–522, 1990.

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

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

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

[9] P. Nesi and T. Querci, "Effort estimation and prediction of object-oriented systems," *The Journal of Systems and Software*, vol. in press, 1998.

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

[11] H. Zuse, "Quality measurement – validation of software metrics," in *Proc. of the 7th International Software Quality Week in San Francisco, QW'94*, pp. 4–T–2, Software Research, 17-20 May 1994.

[12] J. Daly, J. Miller, A. Brooks, M. Roper, and M. Wood, "Issues on the object-oriented paradigm: A questionnarie," tech. rep., Dept. of Computer Science, Univ. of Strahclyde, UK, RR-95-183, June 1995.

metric	comment
AC_m[9]	Attribute Complexity/size
$CACI_m$[9]	Class Attribute Complexity/size Inherited
$CACL_m$[9]	Class Attribute Complexity/size Local
CC_m[9]	Class Complexity/size
CC'_m[9]	Class Complexity/size, predictive form
$CCGI$[21]	Class CoGnitive Index
CI_m[9]	Class Method complexity/size Inherited
CL_m[9]	Class Method complexity/size Local, equivalent to CM_m
CM_m[9]	Class Method complexity/size (pure functional) equivalent to CL_m
$CMICI_m$[9]	Class Method Interface Complexity/size Inherited
$CMICL_m$[9]	Class Method Interface Complexity/size Local
DIT[20]	Deep Inheritance Tree
ECD[21]	External Class Description
Ha[16]	Halstead metric
$HSCC$[10]	Class Complexity by Henderson–Sellers
LOC	number of Lines Of Code
Mc[17]	McCabe ciclomatic Complexity
MC_m[9]	Method Complexity/size
MIC_m[9]	Method Interface Complexity/size
NA	Number of Attributes of a class
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
NCL	Number of CLasses
NKC[22]	Number of Key Classes
NM	Number of Methods of a class
NMI	Number of Methods Inherited of a class
NML	Number of Methods Local of a class
NOC[20]	Number Of Child
NRC[9]	Number of Root Classes in the system class tree
$NSUB$ [21]	Number of SUBclasses of a class
$NSUP$ [21]	Number of SUPerclasses of a class
SC_m[9]	System Complexity/size
$Size2$[6]	Number of class attributes and methods
T_m[18]	Total m-based functional Complexity
$TJCC$ [19]	Class Complexity
WMC [20]	Weighted Methods for Class, equivalent to CL_{Mc} in our notation

Table 3: Glossary of the metrics mentioned in this paper. Metrics with $_m$ parameter are evaluated on the basis of a functional metric selected from: Mc, Ha or LOC; for example: CC_{Mc} Class Complexity/size based on McCabe ciclomatic Complexity.

[13] S. C. Bilow, D. Lea, K. Freburger, and D. deChampeaux, "Workshop on: Processes and metrics for object oriented development," in *Proc. of OOPSLA'93, Conference on Object-Oriented Programming Systems, Languages, and Applications*, (Washington, DC, USA), 26 September-1 October 1993.

[14] V. R. Basili, L. Briand, and W. L. Melo, "A validation of object oriented design metrics as quality indicators," tech. rep., Dept. of Computer Science University of Maryland, UMIACS-TR-95-40, College Park, MD, 20742 USA, April 1995.

[15] C. F. Kemerer, "An empirical validation of software cost estimation models," *Communications of the ACM*, vol. 30, pp. 416–429, May 1987.

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

[17] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 308–320, 1976.

[18] F. Fioravanti and P. Nesi, "Tool for analyzing c++ code (ver.2.0)," tech. rep., Dipartimento di Sistemi e Informatica, Facolta' di Ingegneria, Universita' di Firenze, RT 21/97, Florence, Italy, 1997.

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

[20] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, pp. 476–493, June 1994.

[21] F. Fioravanti, P. Nesi, and S. Perlini, "Assessment of system evolution through characterisation," tech. rep., Submitted to IEEE ICSE98, and also Tech. Rep. Dipartimento di Sistemi e Informatica, Facolta' di Ingegneria, Universita' di Firenze, RT 22/97, Florence, Italy, 1998.

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

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

[24] M. Campanai and P. Nesi, "Supporting object-oriented design with metrics," in *Proc. of the International Conference on Technology of Object-Oriented languages and Systems, TOOLS Europe'94*, (Versailles, France), 7-11 March 1994.

[25] T. P. Hopkins, "Complexity metrics for quality assessment of object oriented design," in *Software Quality Management II, VOL.2 Building Quality into Software* (M. Ross, C. A. Brebbia, G. Slaples, and J. Slapleton, eds.), pp. 467–481, Computational Mechanisms Press, 1994.

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

[27] D. W. HosmerJr and S. Lemeshow, *Applied Logistic Regression*. New York, USA: Jhon Wiley & Sons, 1989.

[28] S. R. Chidamber and C. F. Kemerer, "Towards a metrics suite for object-oriented design," in *Proc. of OOPSLA'91, 6th Conference on Object-Oriented Programming Systems, Languages, and Applications, ACM SIGPLAN NOTICES VOL. 26, N.11*, (Phoenix, USA), October 1991.

Contents