A Validation of the Component-Based Method for Software Size Estimation

José Javier Dolado, Member, IEEE

Abstract—Estimation of software size is a crucial activity among the tasks of software management. Work planning and subsequent estimations of the effort required are made based on the estimate of the size of the software product. Software size can be measured in several ways: Lines of code (LOC) is a common measure and is usually one of the independent variables in equations for estimating effort. There are several methods for estimating the final LOC count of a software system in the early stages. In this article, we report the results of the validation of the component-based method (initially proposed by Verner and Tate) for software sizing. This was done through the analysis of 46 projects involving more than 100,000 LOC of a fourth-generation language. We present several conclusions concerning the predictive capabilities of the method. We observed that the component-based method behaves reasonably, although not as well as expected for “global” methods such as Mark II function points for software size prediction. The main factor observed that affects the performance is the type of component.

Index Terms—Software size estimation, software measurement, function points, software management, linear regression, neural networks, genetic programming.

1 INTRODUCTION

In the area of software management, the prediction of certain variables is key to good planning. Among these variables, the estimation of software size and development effort are the most important, since they drive the whole process of software development. Although there is no consensus about the nature of the relationship between effort and software size, measurement and prediction of the size of the application are by themselves meaningful.

One of the most used measures of software size—lines of code (LOC)—is one of the independent variables used as input to equations for effort estimation. The fact that the LOC are so relevant in the tasks of managing and planning software projects can be attributed to historic (but justified) causes. LOC is one of the representations of the software system that can be easily manipulated by humans. Although other products are managed and used at different stages of the project evolution, the LOC symbolize what the software is meant to accomplish in actual implementation. In the end, software construction involves a multitude of activities resulting in the design of several algorithms. These, in turn, are represented physically as a set of instructions in a document. Other representations such as the documentation of requirements or the binary executable programs serve other purposes. Several algorithmic methods have been proposed to estimate an application’s size in LOC.

Verner and Tate proposed a method for estimating early in the software life-cycle the LOC of a system [27]. This method, described as a component-based method (CBM), sizes individual components or modules first and then adds the component sizes to obtain an estimated system size. The partition in components depends on the environment (type of software) and, therefore, is not fixed. The Verner-Tate approach tries to generalize the division in components through function point analysis.

Verner and Tate size every type of component by examining the characteristics of each type and looking for its predictors of size. They apply regression methods to the independent predictor variables and the dependent variable (size in LOC) to obtain estimation equations. More precisely, their method first tries to explain the size of the components and then selects for estimation only those predictor variables that are available at the corresponding life-cycle phase. Verner and Tate tested the method in two types of systems—business systems and systems programming applications—obtaining very good values in the evaluation parameters. They also tested the advantages of distinguishing between types of components as compared to using only one type of component, with better results obtained in the former situation.

The Verner-Tate study used LOC as a measure of system size, although it was suggested that token counts could also have been used. The CBM method was applied in detail to an implementation of a large data-centered administrative application developed in a fourth-generation language (Application Language Liberator). Validation of the method was also carried out with another business system application using three different environments: Micro Focus Cobol, Advanced Revelation, and Informix-4GL.

The components identified for Informix-4GL included menus, input, and reports/inquiries. The independent variables were the number of choices within the menus and the number of data elements and relations for input and reports/inquiries. The figures reported by Verner-Tate indicate that the three components exhibited good behavior.
The data set comprised 27 cases (five menus, 11 input modules, and 11 reports/inquiries) and the equations were derived by linear regression. The authors pointed out that while a complexity variable could help in the explanation of the equations in some cases, it could not be used for estimation since no associated variable can be used objectively by the estimators.

In their conclusions, Verner and Tate report some of the advantages of the CBM. For instance, while bottom-up size estimation can be accurate, the degree of accuracy depends on how much is known about the system being estimated. However, the CBM, which can be tailored to different environments, can produce accurate estimates without necessitating the completion or near completion of projects. Size estimation can be performed with the initial increments of a system. We can develop the estimation equations from data analyzed in some of the modules and use these equations to estimate the size of later increments. This flexibility allows us to estimate without having built a large database.

Some of Tate and Verner’s early ideas [24], [25] that led to the CBM have been validated in the work of Cockcroft [2]. Cockcroft tested the relationship between products of the early system specification and the lines generated by a computer-aided systems engineering (CASE) tool. Basically, the work consisted of tracing the early products generated in 12 projects developed with the same CASE tool and finding the equations (by multiple linear regression) to estimate the final LOC count. The results of the estimations were good, although the evaluation of the estimation equations was done on the same projects used for model building. This estimation process falls more in line with a global method than with the CBM, since a set of projects is needed in order to estimate. No estimation can be made within a project.

The approach most similar to the CBM—a model proposed by Hakuta et al. tries to be general and independent of the domain of application [10]. This method can be applied at the program level and takes into account the processing units, processing complexity, and environmental factors. The estimator can use products generated at different stages of development, computing the final size of a program from the sum of its constituents; thus, the approach can be defined as a bottom-up method. This model has been validated in 15 programs. The authors advise standardizing the format and content of design documents in order to estimate efficiently. The use of correction factors is left to the estimators. The model differs from the CBM mainly in that Hakuta et al. use subjective variables such as complexity explicitly for prediction whereas the CBM uses them for explanation (although indicating the possibility of using them for prediction too).

In a similar vein, Humphrey has suggested a “proxy-based” estimating method in which estimators use objects as proxies [12]. The objects are then typified so that a size can be estimated for each category. The language used was C++, but Humphrey’s approach is similar to the CBM because it also typifies components according to the environment. The primary difference is that Humphrey’s method sizes each component by analogy with a database of previous developments. Also, validation of this approach has been very limited.

The CBM allows estimators to consider the specific characteristics within a project while maintaining diversity in the elements that constitute the independent variables for estimation. The CBM is appealing since its strength lies in its ability to adapt to the environment. Its flexibility represents a new paradigm in size estimation. The empirical validation of the proposal challenges the methods by which the partition of the components is fixed (such as function points). The CBM has been used to study the validity of the estimation of incremental developments. This article’s focus is the study of possible variations when working in similar projects but with different groups.

The data used in the present study is described in Section 2, and Section 3 introduces the methods of analysis of the data. Section 4 contains an analysis of the LOC-NOC relationship, since components are the building blocks of the CBM. In Section 5, the predictive capabilities of the CBM are studied, with the intention of improving the results for clusters. Section 6 presents the results of the estimation analysis made by Mark II function points and compares them to those obtained previously. Finally, the conclusions of the study are reported in Section 7.

2 Collection and Description of the Data

The validation of the method proposed by Verner and Tate presented here is restricted to a specific environment such as a fourth-generation language in business applications (Informix-4GL). The experiment consists of reproducing the same software measurement and engineering tasks across several projects. The fact that all the projects have been developed in the same setting allows us to present comparative conclusions to the extent to which variations in predictions could be found. Moreover, the purpose here is not to estimate the size of a system in increments but to compare different estimations made by the CBM across a set of projects. The replication of a method is necessary if we are to assert its general scientific validity; here, replication allows us to have confidence in the size estimations made by the CBM.

Availability of the Informix-4GL environment, in conjunction with a reasonably large sample of developers, has made possible the extensive replication of the method. We have analyzed the results of 46 projects to which the CBM method was applied. The data originate from academic projects in which students developed accounting information systems that have the characteristics of commercial products. Each system, designed for a hypothetical firm, includes some or all of the following subsystems: sales, purchases, inventories, financial statements, and production cycles. Every group had to adapt some of those subsystems to the specific firm proposed, with the types of firms including food distribution companies, car dealerships, subcontract management, toy manufacturers, and others.

In all cases, the accounting (financial statements) had to be implemented; but the student groups implemented the other subsystems according to the objectives of the hypothetical firm. As a result, some groups paid more
attention to the sales subsystem while others focused on purchases or inventories. Furthermore, the reports or other parts of the application that the system had to display had no fixed guidelines. Consequently, the sales subsystem of the food distribution company, for instance, differed from that of the car dealership. Nevertheless, the theoretical foundation of the nature of the activities is equivalent [4]. The financial statements (accounting) subsystem provided the most standardized data input and output since this subsystem included the automation of manual bookkeeping activities. The format of the screens and reports had to conform to standards in accounting practice (ledger, journal, balance sheets, etc.). Some variations were allowed in this case, since modifications to screen and report (e.g., the balance sheet) formats could be presented in different ways, resulting in components of different sizes. No other strict organizational guidelines were provided.

As part of the students’ education, the projects were developed in the final course of the bachelor of science in Informatics program. Although the students were able to choose from a variety of languages to implement the projects, most used Informix-4GL. All the projects that provide data for this article were written in that language. A total of 148 people took part in the projects analyzed here, making up 46 groups: 30 groups of three students, 14 groups of four students, and two students working individually. Almost all the students were 23 years old. Each project was considered a real software project, mimicking the activities that would be carried out in an actual business setting. At the end of the project, a software system was delivered.

In every project, the corresponding systems and software engineering tasks, such as requirements specification, analysis, design, coding, and documentation, were carried out. Tracking and control activities were inherent in those projects. At the end of the project, several metrics were computed. The students received wide exposure to software metrics using the current literature, so that during the project they were able to compute and work with these metrics. However, all the data presented here was computed and reviewed at the end of the project. The method of sizing by components was explained in detail according to the original article and data [27]. At the end of the projects, when the software system was working properly, the students were tasked with 1) measuring the application by counting Mark II function points (and Albrecht function points), 2) counting LOC, and 3) analyzing the data of the project. Effort, although also tracked, is not the concern of this paper. Elsewhere, the other relationships among the variables have been analyzed (not exactly on the same data sets) [6].

With respect to the CBM estimation method, the specific task assigned to the students was simply to divide the LOC of the application into three types of components: menus, input, and reports/inquiries. This division was found adequate and was performed according to Tables 11 and 12 of Verner and Tate’s “A Software Size Model” [27, pp. 272-273]. Since in the present work the purpose is to validate the CBM from the prediction point of view, the selection of predictor variables of the components has been left to the published results, if adequate. Thus, the division is the same as that of Verner and Tate, because they have already defined it for the Informix-4GL and compared it with other languages.

Throughout this paper, the term module is equivalent to component. The LOC for the fourth-generation language were counted—in accordance with Verner and Tate—as physical lines, except for blanks and comments, throughout all types of source files [26]. The students treated nonprocedural code, which basically consists of screen descriptions, no differently than procedural code. Counting LOC as physical lines was easy to implement and produced better results than other criteria such as token counts, as Verner and Tate have described [27, Appendix B].

In the present experiment, the students received no guidelines or clues as to the values obtained in counting the components and data elements or as to how many LOC a component should have. The students simply divided the lines they had coded in the application according to the type of component: menu, output, and input. The counting process was manual and final interviews verified the correctness of that process. Only data judged as reliable are included in this data set, although some mistakes are unavoidable. In this environment, we found that up to 25 percent of the projects could supply data that could generate some doubts (those projects were discarded). The students were provided with the data of the CBM method, but their design and coding activities did not take that data into account in any way, since the partitioning of LOC into components occurred after design and coding were complete.

With respect to partitioning the code into components, some ambiguity appeared in the defining menu components. Some menus could be combined in cascade, often blurring the concept of component. In this case, the criteria used was to decide the closeness to the coherence of the menu, in such a way that a component menu was as coherent as possible from the point of view of the meaning. The validations that could be run when a user selects certain options caused some variations among projects in the number of LOC assigned to a menu.

The students have counted in the components only those LOC that can be “run” or executed by the system. These do not constitute the whole system; there are other, less visible parts. The fact that users cannot see some of the auxiliary programs means that those LOC are not assigned to a component. The same situation applies to the loading and validation of databases, making the final count of system LOC vary by almost 30 percent, in extreme cases. Since the rest of the data relates to the application’s functionality, no prediction equations were tried for those utilities. The LOC reported here represent the sum of the lines included in the component types mentioned. In Verner and Tate’s article, it was assumed that 10 percent was the “extra-LOC” added to those lines already counted in the components.

An exception has been made in the partition of the application. The production subsystem was composed of large modules (usually only one module of about 1,000 LOC) that implemented MRP algorithms. This
subsystem is conceptually different from the rest, as its objective is not an intensive data-oriented subsystem; therefore, we omitted the production portion of the application from the counting and analysis processes.

The students recorded data on predesigned forms with fields for the number of menu options and the number of data elements and relations for input and reports/inquiries. Other formatted sheets were used to record Mark II function points. At the end of the project, the data was reviewed and presented on a sheet indicating the number of the component and the data associated with it. The last task assigned was to apply and to evaluate the equations given by Verner and Tate for the Informix-4GL environment and, optionally, construct and compare another regression model. Some questions arose when we obtained results from Verner and Tate’s original equations [27] that did not correspond with the data recorded. We interpreted these questions as an indication that each project had to develop its own equations. Verner and Tate used the same equations for explanation and for estimation, implying that the variables used for prediction can be used for explanation.

The total number of 117,000 LOC analyzed did not include auxiliary programs that were not assigned as components. The total number of components analyzed was 1,537. Table 1 shows a summary of the data used in this study. The reports/inquiries component shows the highest mean values in all categories. Reports comprised, on average, 46 percent of the LOC of a project, with menus containing approximately 10 percent. On average, a project had 9.2 menus, 11 input modules, and 13.2 reports, reflecting an average total of 33.4 modules per project. These projects can be classified as small or medium by the number of LOC of a fourth-generation language and by the number of Mark II function points.

### 3 Methods of Analysis

Given a set of data from which some equations for prediction are to be inferred, we have basically two ways to proceed: 1) take all cases and derive a model by tuning it, eliminating the cases that seem to distort the true relationships between the variables; or 2) if the number of cases is sufficient, develop a model in a sample (from 60 to 67 percent) and then validate the model in the other subset (holdout). Ideally, if the number of cases is large enough, three subsets of the data should be used: one for model building, the second for model tuning, and the third for giving the true evaluations of the equations.

These two methods are not incompatible, but their evaluations of the predictions of a model are different. In the first procedure, the evaluation can only be made in the same data set; and in the latter process, the evaluation takes into account only the validation data set. The second option is the path followed in this work.

In order to ensure the assessment of the CBM, we have analyzed the data in three ways, with each method of analysis representing a different paradigm in the spectrum of methods. Each of the relationships under study was examined using classical statistical analysis (linear regression models), neural networks, and genetic programming.

The benefits of using different methods for analysis have already been expounded in the work of Gray and MacDonell, which compares a variety of methods [7]. The main benefit is the confidence obtained in the data, given the multiple sources of ambiguity and limitations of individual methods. These include sensibility to outliers, sensibility to the sample, nonlinear relationships, and
others. Here we use multiple linear regression (MLR), neural network (NN), and genetic programming (GP). Using different methods of analysis also provides us with a broad view of each method’s possibilities, and the comparison of the results will allow a stronger assessment of the functioning of the CBM.

MLR, one of the most prevalent methods for analysis, requires several conditions for its use and interpretation of results [20]. NN provides a method of deriving models that precludes reasoning about the relationships between the input and output variables, but that makes it possible to recognize some nonlinear relationships that are not easily identifiable [8]. GP, a technique of automatic programming, uses the principles of natural selection [16]. While MLR leaves the task of the discovery of the most appropriate model to the modeler and NN embeds the characteristics learnt on the sample data into the network structure, GP tries to evolve by trial and error an equation relating the variables. Below these techniques are explained in more detail.

The criteria for evaluation are shown in Appendix B. Since we plan to use the CBM for purposes of prediction, the measures primarily considered are PRED(0.25) and MMRE. The other measures have not provided information qualitatively significant enough for us to infer different conclusions. Once a sample of the data is selected, it is maintained across all methods.

When comparing the CBM to global methods for software sizing, we are confronted with the estimation of LOC from two different viewpoints:

1. Prediction of LOC by taking as independent variables those pertaining to the global values of the projects. Therefore, in order to estimate, different projects are needed. No estimation is possible within a project. This viewpoint corresponds to the analysis made in Sections 4 and 6.
2. Prediction of LOC in a project by considering the different predictions obtained in parts (modules or components) of a project. This component-based prediction is analyzed in Section 5.

Comparison of these two methods is difficult; but when the models’ evaluation measures return similar values, we tend to choose the latter option. The component-based viewpoint implies that predictions can be made much earlier (no need for many projects) and are easier to manage, since monitoring of the predictors takes place from the project’s start.

3.1 Statistical Analysis by Multiple Linear Regression

MLR is a technique of classical statistics for model building. The output variable is assumed to be linearly related to the input variables. However, in cases where we suspect that other functions can better fit the data, some transformations are allowed in the variables to achieve a linear model. How well the MLR-built model fits is usually evaluated by means of the $R^2$ (and adjusted $R^2$) and the analysis of variance. Furthermore, when constructing a model, the analysts must study the effect of specific cases (outliers) on the function estimated. Outliers can be identified by means of the Mahalanobis distance, Cook’s distance, and the leverage of the data point. Mahalanobis distance detects unusual values of the independent variables. Cook’s distance, carefully analyzed by Matson et al. measures the impact of a case in the estimates of the parameters [17]. The leverage of a case describes the impact of the dependent variable’s observed value on the prediction of the fitted value [20]. More analysis, such as examination of the linearity and the equality of variance, should be performed to aid the researcher in grasping the precise behavior of the data.

Since building a model for a data set differs from sampling, the strategy followed here is to build a fairly robust model on the sample (66 percent of the data points). The main criteria for validation lies in obtaining acceptable values of PRED(0.25) and MMRE in the validation data set (34 percent). The previously cited criteria of outlier analysis have been calculated and taken into account, but we omitted cases from analysis only if extreme anomalies appeared. The samples have been selected randomly, but it is known that model building can also be dependent on the sample.

3.2 Neural Networks

Artificial neural networks are nets of processing elements that are able to learn the mapping that exists between input and output data [5], [8]. Some types of neural networks are universal approximators that have already been used in the software engineering field [9], [15], [21]. The motivation for using NN here is that the equation relating input and output variables can be left unspecified, allowing us to model possibly unknown relationships to the estimators. The neural networks in this context act as nonlinear regression models. On the negative side, the manipulation of the mapping learned is very limited, since we cannot meaningfully reason about the weights of the processing elements.

The performance of a NN depends on the architecture and parameters of the net. The networks used here for prediction have two layers, with two nodes in the hidden layer (see Appendix C). We tested other structures that included more layers or more neurons, but they tended to overfit the training data. Ultimately, neural networks have worked well here in approximating nonlinear data, whereas MLR has failed to detect the relationships.

We have also used the NN paradigm to explore clusters of similar data points. In this instance, the NN serves as a companion method to the hierarchical clustering methods of classical statistics. Competitive learning neural networks (CLNNs) are a type of self-organizing NN in which each neuron learns to recognize groups of similar input data (Section 5.3). (See works by Hagan et al. for a detailed explanation of this type of NN [8], [5].) The modeler has to specify the number of neurons of the NN and this corresponds to the number of groups in which the data will be classified (in this sense, CLNNs are similar to hierarchical clustering). CLNNs allow the visualization of the center or neuron of each cluster, which helps in selecting the appropriate number of clusters.
3.3 Genetic Programming

GP, an extension of the genetic algorithms technique, originated after the work of Koza [16]. It is included in the set of methods named *evolutionary computation* techniques [1], characterized by solutions that are achieved through generations of candidate solutions that are pruned by “survival of the fittest” criteria.

The name GP comes from similarities to the biological paradigm of natural selection, in which, schematically, populations of species evolve according to random mutations of the genes and to the species’ appropriateness to its environment. In the case of system identification (symbolic regression), the idea is also simple: generate randomly a set of initial equations that relate the input with the output variable and select the equations according to the principle “survival of the fittest.” The process repeats until an acceptable solution is obtained. Appendix D offers a brief explanation of this method.

GP has been used in a variety of fields, including the automated synthesis of circuits, nonlinear system identification in chemical process engineering (identifying relevant variables), symbolic regression, and regression [19], [28], [11]. Here, we have used genetic programming as an automated symbolic regression to derive the equations.

A nonparametric method, GP does not involve assumptions about the distribution of the data and instead derives the equations according to fitted values only. As used here, GP can be considered an alternative to MLR, allowing us to compare the linear equations with those derived automatically. As applied to the CBM, GP will allow us to observe the nonlinearities that can occur in the application of the method. Since different sets of equations are derived from various runs, only those equations that give the best results in the evaluation data are reported. Parameters of GP have been set heuristically.

For all cases subject to exploration in this work, the GP algorithm has been run a minimum of 15 times. The equation that gives the best fit in the last generation is applied to the evaluation data. In some instances, when two equations from different runs were very close in evaluation, we selected the simpler and clearer of the two. Each run had an initial population of 25 randomly generated equations. The number of generations in each run varied, but the best results were obtained with three to five generations (works reported in the literature use a higher number of generations). Longer generations had a tendency toward larger equations that overfitted the data, giving poorer evaluations. Also, longer generations improved the fitness values very little. The number of equations that remained from one generation to the next represented the 10 percent of the previous iteration. The new equations of the new generation were formed by 1) crossover, in which two equations exchange parts, preserving the syntax of the mathematical expression; and 2) mutation, in which a term of the equation (a function, variable, or constant) is changed randomly.

The final impression is that GP has worked very well with the data used in this study. The equations have provided similar or better values than the regression equations. Furthermore, the equations are “intelligible,” providing confidence in the results.

4 RELATIONSHIPS BETWEEN LOC AND NOC

This initial analysis tests several hypotheses concerning some key variables of the method. The concept of component is a primary element of the method, so we now examine, among other things, whether any relationship exists between the number of components (NOC) of a system and its size in LOC. This is an analysis of what the CBM can devise collaterally, since the variable NOC is a consequence of its application.

Firstly, the relationship of lines of code and number of components (LOC-NOC) is considered; secondly, we appraise the relationship between lines of code and the number of each of the three component types (LOC-3NOC).

4.1 Fitting a Model LOC-NOC

As stated above, approximately two-thirds of the data for model building is used. The model that best fitted the data was the power (log-log) model, $LOC = 104.81 \cdot NOC^{0.687319}$ ($R^2 = 0.64$), that is plotted in Fig. 1. Examination of the data points did not show any special peculiarity under the criteria to fulfill according to the parameters of Section 3.1. Nevertheless, it is evident that more points would be needed in the range of $NOC \geq 60$ for a better definition of the curve. An exponent model also fits the data but with a poorer evaluation.

The null hypothesis that no relationship exists between LOC and NOC is rejected with the t-test having a value of $t = 6.435$ ($p = 0.000$). Using the whole data set for the sake of assurance, we obtained a similar equation, $LOC = 100.26 \cdot NOC^{0.903863}$, that supported the shape of the line obtained in the sample. Yet, to maintain consistency throughout the work, the evaluation data of Table 2 corresponds to the equation developed with the sample. GP constructs a model with the equation $LOC = 84.16 \cdot NOC - 0.196 NOC^2$ that behaves similarly to the power model. The NN method does not improve these models.

From the predictive point of view, we observed that the values are neither very good nor they can be considered bad. GP and MLR seem to work equivalently. A small improvement is achieved with the equations that express the relationship LOC-3NOC, explained next.

4.2 Relationship LOC-3NOC

In examining the LOC-3NOC relationship, MLR constructs the equation

$$LOC = 115.68 + 75.25 \cdot menus + 73.29 \cdot inputs + 68.42 \cdot reports$$

($R^2 = 0.49$), and GP finds exactly the same equation. A small improvement in the predictions appears.

4.3 Modularity

The data show a positive relationship between the NOC and LOC. With this pattern of increase in the NOC and LOC, we suspect that the size of the module or its inverse (the percentage of components per 100 LOC) could decrease. This equates to a system’s components growing in size as the system grows. We name the percentage as level of modularization, since it gives us an idea of the
relative amount of modules defined with respect to the amount of LOC. This is clearly observed in Fig. 2, where the decrease in the NOC per 100 LOC is obvious as the system increases its size in LOC (Fig. 2 is drawn this way for the sake of clarity).

The null hypothesis that the level of modularization is independent of the project's size is rejected with $t = -3.512 (p = 0.001); \sigma_{\text{Pearson}}(\text{level of modularization}, \text{LOC}) = -0.4679 (p = 0.001)$. Thus, we can expect smaller increases in number of components as systems increase in LOC.

Although the number of components and their size were computed at the end of the project, the decreasing aspect cannot be attributed to the human counting process. The NOC variable exhibited a high correlation with another variable that could be considered more independent of the counting process, such as the number of logical transactions (see Section 6). This eliminates the possibility of variations due to criteria used by different people in different groups and allows us to consider it as a characteristic proper of this type of system.

Fig. 3 disaggregates the data of Fig. 2 by type of component. The decreasing trends of the number of modules per 100 LOC persist in the three cases, with notable accentuation in the case of menus. This emphasis indicates that the increase of size in menus is not as great as in the other two types. Also, the increase of total LOC within menu components is much lower than in the other cases. The lines drawn represent the regression lines for each type of component. The length of the line in the x-axis shows the range of values that have appeared. The null hypothesis that the level of modularization is independent of the project's size is rejected for the three types of components, but the low value of the $R^2$ indicates great variations in modularity.

The basic trends of the CBM with the data used here are as follows:

1. The number of components increases with the size of the application.
2. The level of modularization decreases as the size of the application increases.
3. When dissecting the LOC-NOC relationship by type of component, the menus exhibit a fall much more pronounced than input modules and reports/inquiries. These last two elements (input and reports/inquiries) behave similarly as the global data, although the explanation coefficient $R^2$ is not very high.
4. The evaluation of the predictive capability of the models LOC-NOC and LOC-3NOC is not bad. The latter is more homogeneous across methods. More data would be needed to symbolize precisely the relationship for higher numbers of components.

### 5 Predictive Capabilities of the Component-Based Method

One of the assertions stated originally by Verner and Tate is that the CBM obtains powerful predictions across types of applications [27]. They also compared the predictions with...
those in which the components were not differentiated, and a global estimation of the system as a whole was made. It appeared that the CBM gave better results. In the case of an administrative system, if only one type of component was taken into account, the results were rather worse at the component level and slightly worse at the system level.

Here, the analysis of the CBM in the set of projects is made from three points of view: the first evaluates the method by building a database with all components; the second dissects the data by project and type of component; and the final approach employs strategies for clustering projects to determine the feasibility and desirable properties of building a database.

5.1 Grouping All Components by Type
Under this condition, a database is built by classifying all components obtained, ignoring the group of origin. Thus, the database contains 425 menus, 504 input modules, and 609 reports/inquiries. As usual, a model is derived with two-thirds of the data (using the same groups as in the previous section). The evaluation on the holdout data is very poor, with the best results found being \( \text{PRED}(0.25) = 19 \) percent and \( \text{MMRE} = 0.76 \) for menus, \( \text{PRED}(0.25) = 37 \) percent and \( \text{MMRE} = 0.64 \) for input modules, and \( \text{PRED}(0.25) = 39 \) percent and \( \text{MMRE} = 0.74 \) for reports/inquiries. Thus, amalgamating all cases generated in different groups does not constitute a good approach if strong prediction models are required.

5.2 The CBM in Each Project
The foremost difficulty in applying the CBM to each project is that some projects lack a sufficient number of components to derive a model from a sample and evaluate it in the rest. If we plan to apply the method to every project, our only solution entails using the same data for model building and evaluation. Although this can distort the evaluation of

Fig. 2. Dispersion of level of modularization by size in LOC.

Fig. 3. Disaggregation of Fig. 2 by type of component.
results, it is the only way to present the results of applying
the CBM to each project.

Fig. 4 plots the cases that fulfill the property of “good
project” (PRED(0.25) ≥ 0.75 and MMRE ≤ 0.25) when
MLR is applied. The number of projects with only one
component fulfilling both conditions is 27 (58.7 percent);
14 projects (30.4 percent) had two components fulfilling
conditions; and only two projects (4.3 percent) had all
three components fulfilling both conditions. There were
three projects (6.5 percent) that had no type of component
fulfilling conditions. The two groups with a good
behavior have a small size. From Fig. 4, we can visually
ascertain that groups with two hits fall within a range
smaller in size than groups with one hit. No special
pattern is observed in the three groups with no hits.

Validation of the models on the same data from which
they have been built tends to produce overestimated
results. We obtained a more realistic and better comparative
view when selecting the six cases with largest NOC for each
type of component, allowing us to define a sample and a
holdout for validation. The results of evaluation are shown
in Table 3. The best evaluation for each case appears in
boldface. Menus perform perfectly, but the equations are
not interchangeable among cases. For instance, the equation
for M1 is $LOC = 2.533 + 2.576 \cdot choices$; for M2,

\[
LOC = 43.68 - 0.4344 \cdot choices;
\]

and for M3, $LOC = 7.092 + 1.794 \cdot choices$. Input modules
behave poorly in two cases (I5 and I6). A low MMRE and a
moderately good PRED(0.25) is obtained in the rest. The equation for I1 is:

\[
LOC = 71.49 - (22.8/data elements) - 13.4 \cdot relations (GP).
\]

Reports also have two cases with poor behavior (R2 and
R5); the other cases approach the limit of being considered
good. MLR provides the equation

\[
\log(LOC) = 1.7 + 0.41 \cdot relations + 0.14 \cdot data elements
\]

for case R6, and GP generates

\[
LOC = 50.7 + 1.501 \cdot data elements
+ data elements \cdot relations - 0.5581 \cdot relations.
\]

In the case of linear relationships, some of the small
improvements obtained by GP compared to MLR come at
the expense of the simplicity of the equations, but the
majority of the linear equations are rediscovered by GP. NN
shows its value in working with nonlinearities, as evidence-
denced in the cases of input modules and reports. In fact,
MLR only approaches the achievements of NN and GP in a
few specific cases.

In terms of the general evaluation of the CBM, menus
have a superior performance to the other two component
types, the relationships being linear. Input modules and
reports have nonlinear relationships relating LOC to
predictors. This can be noticed in the superior performance
of NN and GP compared to MLR. The predictive values of
input modules and reports are similar to or better than
those obtained for the global methods stated in Sections 4
and 6. The MMRE is near or below 0.25 in many cases, and
the PRED(0.25), although not very good, maintains accep-
table levels compared to LOC-NOC and LOC-MKII models.

We obtained improvements when we eliminated some
extreme points in certain groups. Therefore, the figures
reported in Table 3 can be considered as the values for
starting improvement at the project level. Since this only
affects specific projects, the next subsection tackles the
questions from a global point of view. It has been proved
that the equations of one group did not fit into the data of
the other groups, meaning that the equations are character-
istic of each project.

The data reported by Verner and Tate [27] behaves
better in the input and report/inquiry components than
here (Menus: $R^2 = 0.92$, MMRE = 0.053, PRED(0.25) = 1.00,
F-ratio = 33.4; Inputs: $R^2 = 0.59$, MMRE = 0.108, PRED(0.25) =
1.00, F-ratio = 5.86; Reports/inquiries: $R^2 = 0.67$, MMRE =
0.093, PRED(0.25) = 0.91, F-ratio = 8.19). Also, from these
figures, a small difference is noticed in the case of menus compared to the other two component types. It has not been possible (generally) to apply the precise equations given by Verner and Tate [27].

5.3 Clustering Projects

Some of the previous results suggest that improvements could be made in the CBM in some types of components. Several strategies have been used to test whether the method could be improved ex post.

The line of research followed here has used hierarchical clustering (HC) and CLNNs for finding groups of projects that could improve the results. HC consists of combining cases and computing a distance that reflects the closeness of the cases in the new cluster. The euclidean distance is used to measure closeness between cases. Starting from two cases, new clusters are formed. The clustering process continues until all the cases are in a unique group. This process is called agglomerative hierarchical clustering. The dendogram, a diagram that represents the cases or clusters that are combined at each step, also indicates how far or close the cases are in the new cluster. The work of Hutchens and Basili provides examples of how clustering and dendograms have already been applied in the software engineering field [13]. The dendogram aids the user in choosing the number of clusters and cases for each cluster. CLNNs are valuable in helping to determine the appropriate number of clusters. While HC generates a global view of the process of clustering, CLNNs facilitate selection of the adequate distribution of clusters.

In this way, many hypotheses have been examined: clustering by dependent and independent variables, examination of patterns of PRED(0.25) and MMRE, examination of the level of modularity, size, and others. Some of these attempts have generated relative success individually but are not sufficiently conclusive to give definitive criteria for improvement. For instance, correlation was too low to statistically assert that size, per se, is a factor with respect to the magnitude of the relative error and to the prediction level.

Some significant improvements appeared when we jointly considered low size and similar level of modularization. In this instance, we concluded that without a high coherence among projects, we would have difficulty improving the results. In the case of large variations in the parameters of modularization (mean and standard deviation of LOC per component), the process of clustering will not give better results. Clustering of projects could be a correct choice if the cases exhibit some common properties with respect to the mean and standard deviation of LOC of a component type. Mean values hide large variations among the cases of a group.

The task is to find some external characteristic at the project or component level that would allow us to group projects for estimation. This is difficult because there is a clear heterogeneity among groups in the way the systems were designed, which translated into variations in the number of LOC in the components and variations in the rest of the data such as the number of options, data elements, and relations.

The essence of the estimation (and explanation) process is that the predicted variable must have some type of objective relationship with the independent variables across all the data points. In order to depict graphically the relationship between the variables, we can define a simple variable such as “sum of the values of the predictors.” Then we can define the quotient LOC/∑(values of the predictors). If the value of this ratio is constant across a set of components, then we can infer predictions that have the desired evaluation integrity. However, components of the

<table>
<thead>
<tr>
<th></th>
<th>Linear Regression</th>
<th>Neural Networks</th>
<th>Genetic programming</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PRED(0.25) %</td>
<td>MMRE</td>
<td>PRED(0.25) %</td>
</tr>
<tr>
<td>Menus M1 (15)</td>
<td>100 0.03</td>
<td>100 0.05</td>
<td>100 0.02</td>
</tr>
<tr>
<td>M2 (26)</td>
<td>89 0.18</td>
<td>89 0.13</td>
<td>89 0.18</td>
</tr>
<tr>
<td>M3 (18)</td>
<td>100 0.04</td>
<td>100 0.05</td>
<td>100 0.04</td>
</tr>
<tr>
<td>M4 (19)</td>
<td>100 0.10</td>
<td>100 0.07</td>
<td>100 0.07</td>
</tr>
<tr>
<td>M5 (18)</td>
<td>100 0.08</td>
<td>100 0.12</td>
<td>100 0.15</td>
</tr>
<tr>
<td>M6 (15)</td>
<td>100 0.06</td>
<td>100 0.09</td>
<td>100 0.06</td>
</tr>
<tr>
<td>Inputs I1 (25)</td>
<td>50 0.25</td>
<td>50 0.26</td>
<td>62 0.21</td>
</tr>
<tr>
<td>I2 (21)</td>
<td>57 0.24</td>
<td>57 0.32</td>
<td>57 0.23</td>
</tr>
<tr>
<td>I3 (29)</td>
<td>20 0.59</td>
<td>30 0.60</td>
<td>20 0.64</td>
</tr>
<tr>
<td>I4 (17)</td>
<td>33 0.27</td>
<td>100 0.09</td>
<td>100 0.10</td>
</tr>
<tr>
<td>I5 (32)</td>
<td>60 0.34</td>
<td>60 0.26</td>
<td>50 0.30</td>
</tr>
<tr>
<td>I6 (20)</td>
<td>14 0.41</td>
<td>43 0.44</td>
<td>43 0.38</td>
</tr>
<tr>
<td>Reports/Inquiries</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1 (43)</td>
<td>36 0.36</td>
<td>57 0.29</td>
<td>36 0.32</td>
</tr>
<tr>
<td>R2 (24)</td>
<td>38 0.58</td>
<td>50 0.51</td>
<td>37 0.49</td>
</tr>
<tr>
<td>R3 (30)</td>
<td>60 0.23</td>
<td>60 0.20</td>
<td>60 0.22</td>
</tr>
<tr>
<td>R4 (27)</td>
<td>56 0.29</td>
<td>67 0.23</td>
<td>67 0.30</td>
</tr>
<tr>
<td>R5 (35)</td>
<td>25 0.49</td>
<td>50 0.54</td>
<td>33 0.58</td>
</tr>
<tr>
<td>R6 (20)</td>
<td>71 0.29</td>
<td>71 0.25</td>
<td>71 0.24</td>
</tr>
</tbody>
</table>

* The number of components for each case appears in parentheses. Note: The best evaluation for each case appears in boldface.
projects show large variations in this desirable property, which limits the possibility of grouping.

Fig. 5 graphically depicts this idea. For the input component, the relationship LOC versus the quotient previously defined is plotted for a range of values. The shaded area represents a set of components with the quotient ranging from six to 14. This range includes 221 (44 percent of the total) input components (some points are out of the range of the axis), and the evaluation gives values of PRED(0.25) = 63 percent and MMRE = 0.24. Results of validation always improve as the horizontal band is made thinner. So, by maintaining the quotient within an appropriate range, it is possible to group components of different projects for reasonable estimations allowing conditions PRED(0.25) and MMRE to exist. The large dots, which correspond to the values given by group I5 (Table 3), enable us to see how a specific project behaves in this plot. Thus, if all the components of different groups are within similar ranges, then clustering can maintain the estimation property.

In the set of components of a project, some cases distort the estimation property because of, for instance, greater manipulation among relations, increasing the number of LOC without increasing the count of predictors. When we consider projects jointly, only homogeneity can result in ability to estimate. The final conclusion is that it is difficult to relate individual project-level characteristics (such as modularity, number of transactions) to the ability to estimate.

5.4 Summary

Up to now, we have seen that building a database without differentiating projects is not an appropriate choice for improving the integrity of evaluations. No special pattern in variables has been found for the cases that can be classified as good; nor can a pattern be found among level of modularization, size of component, level of prediction, and magnitude of the relative error. The initial evaluation of the CBM gives almost ideal values for menus and moderately good values for input modules and reports/inquiries. We can improve the integrity of these values by eliminating the extreme cases. The values are reasonably compared to those provided by global methods (Sections 4 and 6), taking into account that no previous agreement was set with the developers for module definition. Clustering the groups by close similarity among them is the process that would best improve the values.

6 Predictive Capabilities of a Global Method (Mark II) and Comparison

This section compares the CBM with what could be called a global method—meaning that size predictions are made by sizing the whole project and not by predicting the size of a project’s parts. Thus, the prediction process consists of finding the equation (through the analysis of a set of projects) that expands the attributes that act as predictors into the final desired measure (here, LOC). This is conceptually different, and opposite in several aspects, to the idea proposed by Verner and Tate. However, the final goal is also to estimate the final size in LOC, using as predictors some measures (values) obtained in the first stages of application development. More specifically, the approach taken here is to use as a predictor some global measure of the system, such as function points. Then we try to predict the LOC based on an equation obtained by analyzing pairs of these measures in a set of projects.

In the analysis that follows, Mark II (MKII) function points for 42 (out of 46 projects) were computed following the definitions given by Symons [22], [23], (see Appendix A). Although Albrecht function points were also computed, they were not revised and have not been taken into account. We based this decision on some practical deficiencies detected and related to the definition of logical files in a fourth-generation language environment. The function points reported here are unadjusted function points (UFP
The technical complexity adjustment (TCF) for our environment was set at 16. Calculating the final count does not make any difference for the results presented here.

Since the model considered in Section 4 is also a global model, part of its connection with the LOC-MKII can be examined with the relation NOC-number of transactions MKII. As can be seen in Fig. 6, a strong correlation exists between number of transactions and the number of components with $\sigma_{\text{Pearson}} = 0.816 \ (p = 0.000)$, and the $R^2$ is 66.7 percent ($t$-test = 8.939, $p = 0.0000$). This correlation is neither strange nor coincidental, since many logical transactions can be associated in a natural way with a component. This means that division into components can be prefixed at the specification stage of a system. Thus, transactions such as “add new customer,” “visualize purchase order,” or “report monthly sales” are, in the last instance, events with meaning for the final user. It is certain that many of them have association with only one component at the design and coding phases.

The MKII method divides the system from the viewpoint of the logical actions of the user. In many cases, the transaction-component association is immediate. For example, “add new vendor” allows us to define an input type of component (or module) in the system. In the same way, “generate report xyz” defines a report component, and similar associations can be made with the rest of the logical transactions. The main point is that the division into components comes naturally from the specification of the system. This eliminates the possibility of thinking of the result as a consequence of human processes, and leads us to believe that the subdivision follows naturally from the characteristics of the system. The association of a logical transaction to a component facilitates the comprehension of the system and the partition into components.

A relationship also exists between the number of transactions and LOC with the following values in the linear regression: $\sigma_{\text{Pearson}} = 0.7164, \ p = 0.000; \ R^2 = 51.3 \text{ percent } (t$-test = 6.575, $p = 0.000)$.

The most important relationship to examine in order to compare both methods is the pair LOC-MKII; the plot is shown in Fig. 7. In that figure, the plot of LOC against UFP MKII for every project (43) and the lines fitted are shown. The correlation between variables is $\sigma_{\text{Pearson}} = 0.812 \ (p = 0.0000), \ R^2$ is 0.659, and adjusted $R^2 = 0.6509$. The equation that relates the two variables is

$$LOC = 916.64 + e^{0.0049 \cdot UFP_{MKII}}.$$

GP generates the equation

$$LOC = 729.98 + 0.6966 \cdot UFP_{MKII} \cdot \sqrt{UFP_{MKII}}.$$

The evaluation of the model is shown in Table 4. These values are similar to those of Table 2—especially to the model LOC-3NOC—supporting the hypothesis of the existence of a variety of “function point-like” techniques.

There are other published data concerning the expansion of Albrecht function points into LOC [14], but the results of the evaluation of the predictive capabilities in fourth-generation languages are not known. Moreover, Albrecht function points and MKII do not have a perfect correlation, and in the present case the correlation of LOC and MKII is a little lower than presented in Dolado’s work [6], where a

<table>
<thead>
<tr>
<th>TABLE 4</th>
<th>Evaluation of the Relationship LOC-MKII</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC-MKII</td>
<td>42 projects</td>
</tr>
<tr>
<td>LOC-MKII</td>
<td>PRED(0.25)%</td>
</tr>
<tr>
<td>67</td>
<td>0.28</td>
</tr>
</tbody>
</table>
Pearson $\rho = 0.7641$ (p = 0.000) was found (not measured on the same data sets). We can attribute this difference to the fact that the present study does not take auxiliary programs into account; this eliminates a source of variation in the amount of LOC. No other studies of LOC and MKII have been found.

Another, rarer hypothesis has been tested for MKII. A linear regression of LOC has been made against the independent variables: 1) the sum of input data elements counted in all the transactions of a project, 2) the sum of all the entities referenced and counted in all the transactions of a project, and 3) the sum of all report/inquiry data elements counted in all transactions of a project. Thus, for a project we have the quadruplets (LOC, $\sum$ input data elements over all logical transactions, $\sum$ entities referenced, $\sum$ report/inquiry data elements) where the $\sum$ adds over all logical transactions, independently of how many occurrences of the same data element appear. The underlying motivation for this is to examine the results when the constituents of the MKII method are taken into account. The best results are for GP (MMRE = 0.27, PRED(0.25) = 67 percent); NN is worse (MMRE = 0.28, PRED(0.25) = 53 percent); and MLR performs poorly, which means that this relationship works similarly to the LOC-MKII. No other benefits are observed.

7 Conclusions

The component-based method basically consists of taking advantage of the qualitative characteristics of the software system’s different parts in order to improve the estimation method’s predictive capabilities. The two basic tenets of the CBM, as defined originally by Verner and Tate, follow:

1. The partition of a system in components is not technology-independent; and function-point techniques are only the de facto measure of functionality.

2. Estimation of the size of a system by sizing the components (bottom-up) can be advantageous and accurate.

The current study corroborates these two assumptions. First, a classical global method (LOC-MKII) can be replaced by other global methods (LOC-NOC or LOC-3NOC) defined for specific environments. The comparative evaluation of the models provides similar results in predictive capabilities (Sections 4 and 6). Second, the predictive evaluations of the models developed for each project and type of component are perfect in the case of menus and moderately good in the other two types (Section 5). The figures can be improved by analyzing each of the models in question. In the worst scenario, the values are comparable to those obtained by global methods. The method works very well in those parts (menus) that have a more defined structure that is less prone to variations introduced by developers. The better results obtained for menus than for input or report/inquiry components is noticeable here. These last two types of components have some inherent characteristics that allow the developers to codify the functionality of the system more freely than in menus. Grouping projects is a more conflictive subject and it is only possible to estimate system size if the components are homogeneous across projects.

Comparison against global methods also has to take into account other nonquantifiable characteristics. Because in one case, we estimate the size of parts of an application and in the other case we estimate global LOC, the inference is that estimating by components is a better choice (even if similar results are attained). Another issue to be researched is the application of the CBM to object-oriented systems, where methods and classes can also be typified. Areas that should be examined include the comparative results of the CBM in very large systems and the extent to which establishing standards could obtain the desired properties for estimation.
The component-based approach is promising, since it generalizes the structure of the function-point methods in addition to quantifying components and predicting LOC. We can conclude that the CBM performs reasonably well and offers more possibilities of control than estimation from MKII or any other global method.

**APPENDIX A**

**THE MARK II METHOD**

Mark II function-point analysis (MKII) was proposed by Symons as a better function-point technique after he noted several shortcomings of Albrecht function points [22], [23]. His proposal consisted of considering a system composed of logical input/process/output transactions. The central concept in this method is that of entity, that replaces the concept of logical file. The formula to compute the size in an unadjusted function-point count (UFP) is \( UFP = N_i W_i + N_r W_r + N_o W_o \), where \( N_i \) is the number of input data element types; \( N_o \) is the number of output data element types; \( N_r \) is the number of entity-type references; and \( N_i, N_o, \) and \( N_r \) are each summed over all transactions.

A logical transaction is a unique input/process/output combination triggered by a unique event of interest to the user or a need to retrieve information. The \( W_i, W_r, \) and \( W_o \) are the industrial weights given, related to the effort spent in every component. The values used in the calculations for those weights are \( W_i = 0.58, W_r = 1.66, \) and \( W_o = 0.26. \) In 1988, the values reported were \( W_i = 0.44, W_r = 1.67, \) and \( W_o = 0.38. \)

Finally, the UFP count is calibrated with a technical complexity factor (TCF) that includes 19 characteristics (as well as other possible characteristics added by the user). The TCF is

\[
TCF = 0.65 + C \cdot (\text{Total 'Degree of Influence'}). 
\]

C is obtained by calibration, and the value provided by Symons is 0.005. The total “degree of influence” is obtained by adding the values assigned (in a 0 to 5 scale) to the 19 application characteristics, plus any client-defined characteristics. Then the final adjusted function-point number is

\[
\text{Function Point Index} = (\text{Total UFPs}) \cdot (\text{TCF})
\]

For our environment, the final formula is

\[
\text{Function Point Index} = (\text{Total UFPs}) \cdot (0.65 + 0.005 \times 16)
\]

\[
= (\text{Total UFPs}) \cdot 0.73.
\]

This means that adjusting the count makes the final value 73 percent of the unadjusted one.

**APPENDIX B**

**EVALUATION CRITERIA**

The evaluation of the goodness can be done according to the examination of the following parameters, although in practice many of them do not provide more information and they are not reported.

- **Mean magnitude of relative error**, MMRE, is defined as,

\[
MMRE = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{e_i - \hat{e}_i}{e_i} \right|
\]

where \( e \) is a real value of a variable in a project, \( \hat{e} \) is its estimate, and \( n \) is the number of projects. Thus, if the MMRE is small, then we have a good set of predictions. A usual criteria for accepting a model as good is that the model has a MMRE \( \leq 0.25 \).

- **Prediction at level l—PRED(l)—**where \( l \) is a percentage, is defined as the quotient of the number of cases in which the estimates are within the \( l \) absolute limit of the actual values divided by the total number of cases. For example, \( \text{PRED}(0.1) = 0.9 \) means that 90 percent of the cases have estimates within the 10 percent range of their actual values. A standard criteria for considering a model as acceptable is \( \text{PRED}(0.25) \geq 0.75. \) This means that at least 75 percent of the estimates are within the range of the 25 percent of the actual values.

- **Coefficient of multiple determination** (\( R^2 \)) and **adjusted \( R^2 \)** are some usual measures in regression analysis, denoting the percentage of variance accounted for by the independent variables used in the regression equations.

- **The mean squared error** \( SE \) and the root mean squared error \( (SE)^{1/2} \) as defined by Conte et al. [3] are

\[
SE = \frac{1}{n} \sum_{i=1}^{n} (e_i - \hat{e}_i)^2
\]

and,

\[
(SE)^{1/2},
\]

respectively; the **relative root mean square error**, RMS, is defined as

\[
RMS = \frac{RMS}{\sum_{i=1}^{n} e_i}.
\]

The \( SE \) differs slightly from the mean squared error obtained in regression models. A model is considered acceptable if \( RMS \leq 0.25. \)

- **Mean squared error**, as defined for regression models [17], is

\[
mse = \frac{1}{n-2} \sum_{i=1}^{n} (e_i - \hat{e}_i)^2.
\]

**APPENDIX C**

**NEURAL NETWORKS**

The software used to implement the neural networks is Matlab’s Neural Network Toolbox [5]. Fig. 8 shows the structure of a feed-forward NN used to learn the relationship between LOC and the independent variables number of data elements and number of relations. The hidden layer is composed of two neurons that simulate a logistic-sigmoidal function.

\[
\text{DOLADO: A VALIDATION OF THE COMPONENT-BASED METHOD FOR SOFTWARE SIZE ESTIMATION}
\]

1019
\( f(n) = 1/(1 + e^{-n}) \). The output layer is a linear transfer function.

To apply the learning algorithms correctly, all data is normalized and then denormalized afterwards for evaluation of the predictions. The inclusion of the maximum and minimum of each variable is a necessary condition for good generalization. The learning algorithms used are the Levenberg-Marquardt and the backpropagation methods, the former being faster while obtaining similar results with the first one. The initial weights and biases of the neurons were chosen randomly. Multiple runs were performed to find the correct settings of the parameters.

**APPENDIX D**

### The Genetic Programming Technique

The software for GP has been provided by the Newcastle Symbolic Optimisation Group (runs in Matlab 4.2c).

The algorithm that evolves a solution for symbolic regression is:

**Genetic Programming Algorithm**

```
Generate initial population (of size N) of equations
While there are generations to run do
   Evaluate fitness of each equation
   For each equation in the population
      select randomly one of
      a. Mutation with probability Pm
      b. Crossover with probability Pc
      c. Direct reproduction with probability
         (1- Pm -Pc)
      Add the new equation to the new population
   endfor
endwhile
```

Each equation in the population is represented as a tree. Fig. 9 represents the equation

\[
LOC = 84.16 \cdot NOC - 0.196 \cdot NOC^2.
\]

The operation of crossover takes a subtree of two equations and exchanges them to form new members of the populations. Mutation randomly modifies a subtree of an equation.

The functional set used is: +, -, *, /, ^, square root, square, log, exp. Probabilities of mutation and crossover are in the range of Pm = 0.8 and Pc = 0.2. The number of generations producing best results varied from three to five. One problem posed by GP is that some limit to the trees has to be set in order to generate simpler expressions. Here, the best results were obtained by limiting the number of generations. Contrary to the results of McKay et al., the best fitness measure in the present work is the mean squared error of the predicted values of the equation, and not the correlation coefficient between the input data and the predicted values of the equation [18]. In the implementation used here, regression constants in the equations are determined using the Levenberg-Marquardt method of nonlinear least-square optimization.

**ACKNOWLEDGMENTS**

The author would like to thank his students and to Luis Fernández Sanz. The analysis by genetic programming has been possible thanks to the software of the Newcastle Symbolic Optimisation Research Group (http://lorien.ncl.ac.uk/sorg), to whom I am very grateful. Finally, three anonymous referees provided invaluable comments that have shaped this work. The data of this work is available at http://www.sc.ehu.es/jiwdocoj/cbm.htm. Part of the work was carried out under project UPV-EHU 141.226-EA 114/96.

**REFERENCES**


José Javier Dolado received both his BSc and PhD in computer science from the University of the Basque Country, Spain in 1985 and 1989, respectively. He is a lecturer in the Department of Computer Languages and Systems at the University of the Basque Country, Spain. He was awarded three prizes for his academic achievements. His current research interests are in software measurement, dynamics of the software development process, qualitative reasoning, and complex systems. His works have appeared in several refereed journals and he has served on various program committee relating to international conferences on software quality and process improvement. He is a member of the ACM, ACM Sigsoft, IEEE, and IEEE Systems, Man and Cybernetics Society.