The Role of Components of Data Flow Diagram in Software Size

Simon, Iok Kuan WU
Faculty of Business Administration
University of Macau
Macau
Tel.: (853) 3974749
Fax: (853) 838320
E-mail: simonwu@umac.mo

Abstract

Managing and estimating a good software is not an easy task. Because software estimation activities are concerned not only with time and effort scheduling, but also with specifying work activities, skill levels and scheduling of necessary resources. With duration, effort and other factors overlooked, poor reliability and functionality of software may occur. Furthermore, inaccurate estimation will lead to high pressure for the working team, and poor quality of final development. Hence, designing a right software metric is an important task. Due to these reasons, a software size model is being developed using a sample consisting of 122 student projects. This model takes several advantages: (1) there tends to be fewer counting problems than other software metrics, because this model is based upon simple counts; (2) the predicted software projects were calibrated to specific local environments rather than being based upon industry weights; (3) basic size components can be identified easily at the early stage of the development life cycle; (4) the model provides clues to project designers in planning and scheduling the development of new information systems.

1. Introduction

Software size models have become quite sophisticated since their introduction during the early 1970s. At that time, software development cost models were only based on a single parameter such as program size. These models were not accurate and reliable, as they were usually developed from a limited database, both in number of programs and a variety of applications. Furthermore, according to Boehm [2], these early models varied widely in their underlying assumptions and definitions of size, so the cost equations differed substantially.

Software size is a key factor to all software cost models, however, as late as the early 1980s, a few size estimation models were developed. Bozoki [4] developed a well-known expert judgment sizing model, a variant of which is now incorporated into the SEER family of models as SEER-SSM. PRICE Systems added a parametric sizing model. This model considers about twenty factors, including size calibration factor to estimate lines of code (LOC). Unfortunately, LOC is only restricted to using one particular language to estimate the software size.

Upon the introduction of Function Points Analysis by Albrecht [1], it is used to measure the functionality delivered by software and a measure about the functionality that the software delivers to the users [7] [13]. An extension of Albrecht method is known as Mark II function points [13]. This model considers a system consisting of some logical transactions. Each transaction consists of inputs that are received, data stores that are referenced, updated and outputs that are generated. Based on the number of transactions in an information system, each transaction is individually sized and the total counts are obtained toward the overall information processing size for the whole system. In both of these two metrics, although significant effort has been devoted to strengthening the counting practices, but questions of subjectivity and measure interdependence remains. Function point method is a complex process that required a degree of training [11] [3]. While Mark II function point uses industry average weights as a measurement for construction. It is unclear how representative the systems contributing to the average are, and how stable the averages are over time. In addition, it is also unclear such weights are appropriate for new systems as well as system enhancements [12].

A similar approach, developed by Verner et al. [15], which is a bottom-up method in which the final counts of software are determined by summing up all individual components. The results of the method were very significant, but the sample data is restricted to only one system. This implies the model may not be generalized to other environments.

2. Research model

The objective of the study is to explore the possibility of developing a software metric using 4GL software projects. The model is simple and easy in estimating software size. It can be applicable to other software projects within the same environment as well. In the study, two sets of project specifications are examined. The first set is user specification, and the second set is program size in terms of executable statements (size).

2.1 User specification

The user specification is Data Flow Diagrams (DFDs) from which the software size components are extracted directly. This approach is totally different from those based on LOC and other software metrics, because the software size is predicted and estimated using simple counts (components) at the early stage of the development life cycle. Based on the details of DFDs, the software size components are extracted directly. Before studying the details of DFDs about the software projects, the context diagram is evaluated first. Walking through the context diagram, it shows how software projects are modeled and

interact with other modules/systems as a whole if any. However, context diagram is not detailed enough for representing the whole project's data movement, because it lays out only the sources/sinks and a single process with a generic name representing the entire software. There are no data stores and other components identified at this level. So DFDs are studied for understanding the entire software project level by level. By studying DFDs, project designers have better understanding about the data movement throughout a business process or an information system that the data undergoes.

From the first level of DFDs, it shows the main processing activities of the entire software project, such as processes are shown and data stores are added at this level. It is probably the most informative level compared to context diagram, because the context diagram gives project designers little information about what the software functions. Before proceeding with an explosion of next level of DFDs, project designers must be aware of the balance of the child diagram against the parent. To do so, they make sure both the 'data in' at the child level is identical with or adds up to the 'data in' at the parent level and that the 'data out' at the child level is identical with or adds up to the 'data out' at the parent level. The data pass between the processes on the child DFDs do not enter into this evaluation because they are internal to the process at the parent level.

After completing the level balancing, project designers explore each process on the balanced diagram into its own DFD. However, input and output requirements remain constant at that level but the data stores and sources are changed. If there are more details about each data process required, the next level explosion is continued by determining the next lower level's process until it cannot be subdivided, or it cannot be broken down to a point where the bubbles have a single input data flow and a single output data flow. This level is considered functionally primitive [6], because it cannot be further partitioned into subordinate components or the lowest levels have been reached [14] [6] [9] [8], so that the partitioning process is complete.

2.1.1 Calculation of software size components

When the entire software project is broken down into the functional primitives, we are in a better position to extract and count the size components. Before the components are counted toward the final size of a software project, it is essential to evaluate all DFDs carefully by determining their correctness. Because numerous errors, omissions, and inconsistencies can occur for several reasons, such as forgetting to include a data flow, making an arrowhead in the wrong direction, incorrect labeling processes and/or data flow, or creating unbalanced decomposition in child diagrams etc. If the error checking procedure is complete, the process of identifying the size components can be started. The followings are the counting procedures for identifying the software size components.

(a) Input entity (in_enty)

The input entity refers to those input screens enter the boundary of an interface of a software project, or those are directly connected to different processes/modules by data flows. An input screen can be used for the creation of a new record, or can be used to change some existing data items from a database, or to delete an existing record. In the study, different functions of input screens are counted toward the size components. For those input screens that may not require any processing logic at all, such as a start menu or a query input screen. They are summed together to represent one input entity.

(b) Data flow (data_flow)

Data flows are those arrows flowing within each functional primitive. These arrows are those arrive at the processes/modules from input entities, or those enter from one process to another, or to data stores, or those leave from data stores to processes, and those depart from processes to output entities. Each occurrence of data flow within all functional primitives is counted toward the size.

(c) Process

Process is a transformation process for manipulating the incoming data and generating output according to the program statements given. The process component is important to each functional primitive, as it reflects the complexity of the module of the software. Thus, the higher the module complexity, the more interactions are required between input and output activities. In each functional primitive, the functions of the process are varied, such as updating the input data, verifying the existence of an entity (record) or preparing the outputs etc. All processes in each functional primitive are counted toward the size.

(d) Data_store

Data stores are the permanent or semi-permanent places for storing transaction data, they include the master files, transaction files, table files or any other semi-permanent files. The functions of the data stores are used either for looking up or updating purpose, depending on the

transaction processing required. If the data stores are used for looking up purpose, the counting of these data stores are summed together for representing only one data store. Since they are not required for transaction/data processing. However, if the files are used for updating purpose, such as changing the data items or deleting a record, then the data stores are counted individually. As they require different logic for transaction processing.

(e) Interaction

The number of interactions counted toward the software size including the data or control information passed between processes. The counting of interaction between two processes can be one way or two ways. In other words, in a functional primitive, a process acts as a calling process, while another one acts as a called process. In between, when a calling process only directs a called process to perform a specific job by passing either data or control information or both at the same time, one way of interaction is counted. Otherwise, when the called process finishes the execution, it is required to pass the processed information to the calling process, two ways of interactions are counted.

(f) Output entity (out_enty)

For each report/output screen that leaves from the boundary of a module/process after processing, it is counted once if different processing logic is required. On the other hand, if both output screen and printed report are the same length and having the same output data items, only one output entity is counted toward the final size. However, the following outputs are summed together as an output entity and counted toward the final size of the software project:

i. A selection of menu screen: a displayed main menu offers several choices of selection are summed together, because they required no processing logic;
ii. A starting output screen: like the initialization of conversational processing which leads the user to execute different jobs;
iii. A query language output: a displayed message/output screen does not require any processing logic before it is retrieved.

After extracting the size components from all functional primitives, they all are added together and put into the equation for running multiple regression. The complete picture of the model using the size components identified above is presented in Figure 1.

2.2 Size

Size is the measure for expressing how large of the software is in terms of complexity. It is considered as a dependent variable. In this study, the size is referred to the total number of source statements run-on lines with a continuation indicator as a single statement. It excludes blank and comment lines of code, and other job control languages.

3. General data characteristics

All software projects in this study comprising the sample were built over a period of ten years by groups of senior students. Every software project was built to satisfy the real requirements of the external clients, normally most of the clients are from small medium enterprises. Before each project's work activity was carried out by a group of students, they were required to use a prototyping process for development, meeting with their clients on around three occasions over nine-week development period. This process is to ensure that the projects being developed would be satisfied and accepted by the clients. The projects addressed transaction processing, data retrieval and reporting, and file maintenance activities performed by the organizations. Under the software process employed, a software proposal outlining the functionality to be delivered was signed off by the client after one week. On system delivery, the client performed an acceptance test and system review. All projects satisfied the requirements of both the clients, as evidenced by the reviews, and the course administrators, as indicated by the marks awarded (although marks varied over the sample). A wide variety of software projects were constructed over the period. In total, more than seventy distinct working software projects were developed and reviewed.

Although the developers were students, they were in the final segment of their third or fourth year degree. Almost all of them completing their degrees went on to hold development positions with three months of the completion of the projects in the sample. More importantly, the software projects developed were functionality sound, providing an actual working solution to an actual client system. The students all had an equivalent amount of previous experience with the tools and the methodology used.

All software projects were implemented using the same tool, the Cogos 4GL Powerhouse. The software projects were all of the same generic class, such as transaction oriented data processing and retrieval applications. This commonality is advantageous in these factors that can be considered as constant in the system analysis, a condition not often encountered in software size research. When

they vary, factors such as these can clearly have an impact on software size. Given the potential contributors may be treated as constant, the degree of confidence adopted in regard to any size relationships supported by the data will consequently be greater.

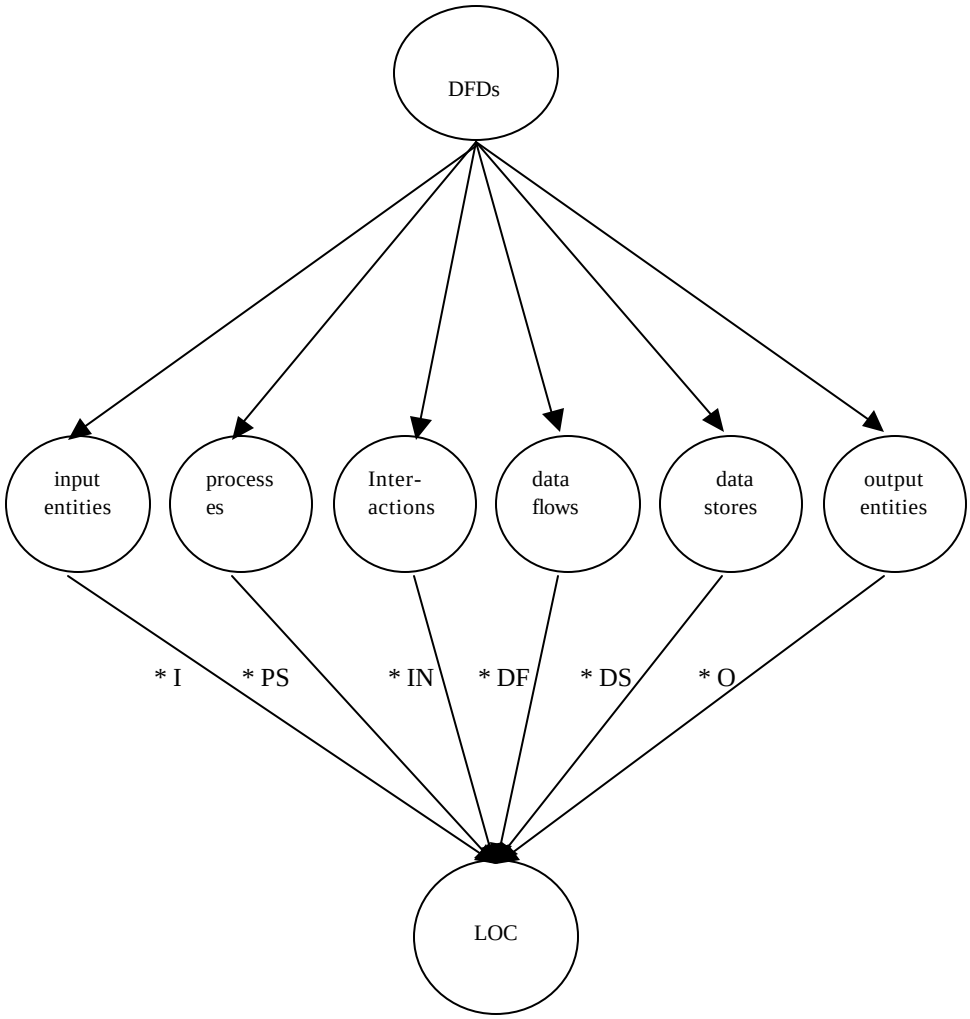

Figure 1- Size estimation using components of DFDs

$$LOC = Wi*I + Wps*PS + Win*IN + Wdf*DF + Wds*DS + Wo*O$$

Where:
Wi is the coefficient of an input entity
I is the number of input entities
Wps is the coefficient of a process
PS is the number of processes
Win is the coefficient of an interaction
In is the number of interactions
Wdf is the coefficient of a data flow
DF is the number of data flows
Wds is the coefficient of a data store
DS is the number of data stores
Wo is the coefficient of an output entity
O is the number of output entities

4. Research methodology

The whole sample of 122 software projects is divided into two subsets of data randomly. The larger subset is called main sample consisting of two third of software projects. This subset is used for primary modeling analysis and prediction. The smaller subset called holdout sample

consisting of one third of software projects. This subset is used for validation of models developed previously using the main sample. In the study, a linear multiple regression analysis is applied for estimating and predicting the software size.

To evaluate the accuracy of regression model, the mean magnitude of relative error (MMRE) and the threshold oriented predicted measure are used. The accuracy of a project estimate is measured using the magnitude of relative error (MRE). The MRE is calculated for a project using the following formula:

$$MRE = 100 \mid (ACT - EST) / ACT \mid$$

where MRE is the magnitude of relative error, EST is the value of estimated size of a software project (in terms of LOC) and ACT is the value of actual size of a software project (in terms of LOC). Thus, the smaller the value of MRE, the better the prediction. But one important thing is that the positive and negative errors will not cancel each other out. So that the larger the value of MRE, the worse the prediction. Therefore, in an attempt to produce on average a good set of predictions, MMRE is calculated using the following formula:

$$MMRE = (1/n) \sum_i MRE_i$$

However, even the MMRE is small, there may be one or more predictions that can be very bad. So a second measure is used to examine the cumulative frequency of MRE for a specific error level. We use the measure called PRED (p) or Prediction at Level p suggested by Conte et al. [5]. If k is the number of projects in a set of n projects whose MRE ≤ p, the PRED (p) = k/p. Following the suggestion of Conte et al. [5], we report the PRED (.25) level for the estimation method. A value of PRED (.25) ≥ 0.75 is considered desirable for size estimation.

5. Correlation analysis

Given the small degree of skewness and influential outliers are identified in some of the independent variables, the remedying steps are performed if necessary. It is essential to assess the independent variables whether they meet the basic assumptions, so as to identify any potentially useful (linear) relationships between dependent variable (size) and independent variables, as well as among independent variables themselves. After remedying the basic violations for some of independent variables, the stepwise multiple regression analysis is performed. The detailed results are presented in Table 1. Examination of the correlation matrix indicating there is a strong correlation results between dependent and independent variables. Since our focus is only to identify general patterns within the entire set of observations, the individual observations should not be omitted as they may strongly influence the regression results. Therefore, we need to identify them and access their impact if necessary.

Table 1 - Correlation coefficients

Variable	Size	In_enty	Out_enty	Interaction	Process	Data_flow	Data_store
Size	1.000	-	-	-	-	-	-
In_enty	.833	1.000	-	-	-	-	-
Out_enty	.596	.491	1.000	-	-	-	-
Interaction	.495	.415	.465	1.000	-	-	-
Process	.735	.574	.415	.310	1.000	-	-
Data_flow	.498	.373	.527	.346	.407	1.000	-
Data_store	.738	.574	.500	.429	.481	.394	1.000

All are significant at 0.01

In Table 2, it indicates the model summary of selected variables before and after removing 11 influential observations. Examination of correlation matrix shows that there is not only close relationship between dependent and independent variables, but also improving both values of adjusted R^2 and the standard error of the estimate. Therefore, we believe the influential observations identified previously have tremendous impact on final regression results. Thus, we accept the selected variables listed in the 'after' column for inclusion in the regression model.

Table 2 - Comparison of model summary of selected variables

Variable	Before			After		
	R^2	Adjusted R^2	Std. Error of the Estimate	R^2	Adjusted R^2	Std. Error of the Estimate
In_enty	.501	.494	213.59	.693	.689	200.43
Process	.757	.750	241.25	.794	.789	195.64
Data_store	.826	.819	230.18	.856	.851	187.73
Out_enty	.845	.836	200.95	.866	.859	170.01

From Table 1 and Table 2, the dependent variable is fully correlated to all independent variables. In other words, the independent variables have varying degree of effect on the dependent variable. Among the independent variables, IN_ENTY is mostly and highly correlated with SIZE. It is firstly considered for inclusion into the regression equation. According to the results presented in Table 2, the adjusted R^2 of IN_ENTY is 0.689 (which is about 69 %). By this value, it explains about 69% of effect of IN_ENTY to the SIZE. This is in the expected direction. Our concern is to estimate the size of software projects as early as possible using known variables. In system analysis stage of the development life cycle, in_enty is relatively easy to identify from each module/process using DFDs. More in_enty implies more time and effort are spent in designing and constructing the software. Therefore, IN_ENTY has a major effect to the size of software projects and it is in the expected direction.

The multiple R value has increased with the addition of the PROCESS variable by 7.2%. While the adjusted R^2 value has increased by 0.1. Indicating with the inclusion of this variable, it adds about 10% to the validity of SIZE which the software project has explained. This is true since the size of software projects can increase accordingly when more PROCESS is involved for processing. At the system analysis phase of the development life cycle, the number of PROCESS could be easily determined using DFDs. In a complete transaction, process is one of the components interacting with input and output. More PROCESS implies higher software complexity and complicated interactions between components of software projects. Therefore, the more the number of PROCESS involved for processing, the higher the complexity of the system interface.

The multiple R value is increased with the addition of DATA_STORE variable by 3.2%. While the adjusted R^2 has increased by 0.062, it implies with the inclusion of this variable, adding 62% to the validity of SIZE which the software has explained. The same result achieved was very significant in a study conducted by Itakura et al. [10]. Its R^2 achieved was 0.8. In another study reported by June et al. [14], the R^2 achieved was 0.98 which is the highest correlation coefficient as compared to other independent variables under the same study.

In every module, the access to data store is necessary. The access to data store could be a simple query, addition, deletion or changing process. The influence of access to data store to size of software projects depends on what kind of function it performs. The influence of access to data store using a simple query would be less compare to the function of adding new records to the same data store. The simple query function does not require higher complex steps in operations, while addition function requires more interactions if more data stores are involved.

Finally, with the inclusion of OUT_ENTY, the multiple R value is increased by 0.3%. While the adjusted R^2 has increased 0.008, it implies that with the consideration of this variable. It adds 0.8% to the validity of SIZE which the software has explained. The explanatory power of this variable is small, it contributes less than 10% to the SIZE. Because producing the details of OUT_ENTY is fully depending on IN_ENTY, PROCESS and DATA_STORE. The output of the report is generated directly by retrieving from DATA_STORE, and the level of complexity required for producing a report relies on the design of process. Therefore, the portion of effect to size of software projects by OUT_ENTY is already explained by both PROCESS and DATA_STORE.

Among the independent variables, DATA_FLOW is the first one to reject due to insignificant effect to the size. As pointed out previously, data flow is used to connect components of DFDs showing the connections between components in functional primitives. After examination of data flow in DFDs, a data flow shows where the next process will be taken place after finishing the previous one, or where the next destination will be for retrieving the data from. Even in a large software project, it does not imply

that more designing effort is required accordingly, but from the occurrence of data flows in the software project, we can only know how the DFDs components are connected. If there are many data flows in the software projects, it means there are more other DFDs components are connected accordingly. However, since data flows don't reflect processing in the functional primitives. Therefore, the variable DATA_FLOW has no impact to the size of software project and is rejected.

For INTERACTION variable, some detailed processing steps may not be known to system designers at the system analysis stage of the development life cycle, especially when comp lexity of software projects is large and complex. At the early stage, we may not be very easy to investigate how DFDs components interact with each other. DFDs only show data movement from one state to another within an entire software project. It would not show the logic or complexity required in a module, even project designers have ideas about how modules interact with one another by passing data or control information from one to another. These ideas could be implemented using other programming techniques, for instance, structure chart or Warnier-Orr Diagram etc. Therefore, the variable INTERACTION is rejected.

With the completed model regression estimation, the regression variate specified, and the diagnostic tests that confirm the appropriateness of the results administered, we can now examine the predictive equation, which includes IN_ENTY, PROCESS, DATA_STORE and OUT_ENTY. The predictive equation is written as follows:

Y = -2008.689 + 37.330 (IN_ENTY) + 253.428 (LOG(PROCESS)) + 207.779 (LOG(DATA_STORE)) + 7.743 (SQRT(OUT_ENTY))

where Y is the dependent variable measured in thousands of LOC, IN_ENTY is the number of input entities, PROCESS is the number of processes after taking logarithms, DATA_STORE is the number of data stores after taking logarithms, and OUT_ENTY is the number of output entities after taking square root. By using this equation, the expected thousands of LOC can be calculated.

6. Validation

After developing the model using the main sample, a validation process is followed using the holdout sample. Before the estimated size of each software project is calculated, the independent variables of the holdout sample are transformed by taking logarithms and other transformation process if necessary.

There are some estimation errors introduced when the software projects were analyzed. The relative errors of these projects are ± 0.2 but with some minor exceptions. It looks like a good model when the mean relative errors (.03) are calculated. A good model will lead to small values of relative errors and generally to a small mean relative error [5]. However, since it is possible that large positive relative errors can be balanced by large negative relative errors, a small mean relative error may not imply that a model is a good one. So the magnitude relative error is computed for each project. Thus, the smaller the value of magnitude relative error, the better the prediction. Even more important, positive and negative relation errors do not balance each other, so that the larger the value of magnitude relative error, the worse the prediction.

In order to produce a good set of project predictions, the mean magnitude relative error (.09) is calculated. The mean magnitude relative error is evaluated using a set of 36 projects whose mean magnitude relative error is ≤ .25. The calculated PRED (.25) =.86. That is to say, in this model, 86% of the predictions for the cases in the set fall within 25% of their actual value. Therefore, this model is being accepted for software size estimation.

7. Conclusion and limitations

The paper studied the effect of the components of DFDs in determining the size of software projects. The analysis indicates the size components are accurate and reliable in predicting the size of software projects if they are extracted at the early stage of the development life cycle. Further, it shows that project managers have better understanding about planning and controlling the software development activities before the software projects are developed.

For project designers, the results of the study provide several insights for future software projects development. (1) the size components are easily extracted from DFDs during system analysis stage. Compared to other software metrics, development effort and duration are minimized without waiting until the entire software is fully developed. (2) the weights are determined after running linear multiple regression using a large sample, as a result, the weights are very reliable for representing the effort devoted to software engineering activities. (3) the model is easier to apply to similar environment. Since the results of the study are calibrated from large sample, the model can be generalized to other similar software projects. (4) the model is very suitable for both small and medium companies, especially for those which are lack of experience project designers and need better management and tighter control at lower cost.

The results in this study appeared promising. However, we should be aware of the following limitations. First, the software projects are small and medium in terms of program complexity. The results of the model are applicable to the similar software projects, such as transaction processing,

data retrieval and reporting generation. Second, the designers in the study are students, the final results of the model may be different if the software projects are developed by the experienced programmers. Third, the model developed in the study specifically emphasized on data-strong software projects, function-strong and hybrid-strong software projects may not be appropriate.

References

[1] Albrecht, A. J. "Measuring application development productivity," *Proceeding of IBM Corp*., 1979, 14-17, Monterey, AC.

[2] Boehm, B. W. *Software engineering economics*, Englewood Cliffs, NJ:Prentice-Hall, 1981.

[3] Bock D. B., & Klepper R. "FP-S: A simplified function point counting method," *The Journal of Systems Software*, 1992,18,245-254.

[4] Bozoki, G. J. "An expert judgment-based software sizing model," *ISPA Journal of Parametrics*. 1993, 2(1), 65-75.

[5] Conte, S. D., Dunsmore, H. E. & Shen, V. Y. *Software engineering metrics and Models*, Menlo Park, CA: Benjamin Cummings, 1986.

[6] DeMarco, T. *Controlling software projects*, Englewood Cliffs, NJ: Yourdon Press, 1982.

[7] Dolado, J. J. "A study of the relationships among Albrecht and Mark II function points, lines of code 4GL and effort," *The Journal of Systems Software*, 1997,37,161-173.

[8] Floyd, C. "A comparative evaluation of system development methods," *Information Systems Design Methodologies: Improving the Practice*, 1986, 19-54.

[9] Gane, C. P. *Data design in structured systems analysis*, In P. Freeman & A. I. Wasserman (Eds.), Tutorial on Software Design Techniques (4th ed.), 1983, 115-132.

[10] Itakura, M. & Takayanagi, A. "A model for estimating program size and its evaluation," *Proceeding of 6th International Conference On Software Engineering*, Japan, 1982, 104-109.

[11] Jeffery, D. R. & Low, G. C. "Function points in the estimation and evaluation of the software processing," *IEEE Trans. on Software Engineering*, 1990, 16,64-71.

[12] Kitchenham, B. A. "The problem with function points," *IEEE*, 1997,29-31.

[13] Symons, C. R. "Function point analysis: difficulties and improvements," *IEEE Trans. on Software Engineering*, 1988, 14(1),2-11.

[14] Symons, C. R. *Software sizing and estimating: MK II FPA*, 1991, John Wiley.

[15] Verner, J., & Tate, G. "A software size model," *IEEE Trans. on Software Engineering*, 1992, 18(4),265-278.