An Experience in Software Measurement in Academic Settings

J.J. Dolado
Facultad de Informática,
Univ. of the Basque Country
20 009 - San Sebastián
Spain
dolado@ si.ehu.es

ABSTRACT.

In this article some experiences related with the design and implementation of experiments in software engineering in an academic setting are reported. The projects in this type of environments can simulate some of the characteristics that arise in real software production environments. It is obvious that data coming from industry is indispensable for advancing the state of software management. Nevertheless, the academic environment allows a stable environment for replication of activities. Specifically in the investigations reported here several relevant questions about function points, effort and software size estimation were tackled. In the present study it was found that effort had a greater correlation to lines of code than to function points. The component-based method for software size estimation was also validated.

I. INTRODUCTION.

Recent international trends tend to encourage experiments of software measurement mainly in the real software production environment. There is no doubt that an approach is desperately needed in order to validate the assumed software practices. Some of the data thus obtained is initially more meaningful to software practitioners. However, we contend that academic environments allow us to control variables more easily than in some industrial environments. Also, the university environment could guide large-scale experiments and can suggest new ideas about the software process. Here we present two examples of how software measurement has been carried out in a university setting.

Since many of the novel methods are only taught at universities and are spread more slowly into the real software industry, their suitability to the software process should be proved at the first tries. Validating the practices we teach with our students and projects will give us more faith in the methods used nowadays. Moreover, academic software projects can be small to medium in size, and could mimic some real software projects. Not all the characteristics can be equally translated into the academic environment. Those that are related with large scale software development, can only be reflected in simplified ways. But many of the elementary tasks that compound such large developments can be tested or verified in small-scale experiments.

In the present experiments, we have attacked some questions related with software cost estimation. Some well-known cost estimation models can not be applied in many small or medium real industrial software projects, and hide the problems that still pervade the software engineering process. Most of our students will not be in the future in charge of very large projects, but they will be developing or managing small or medium projects. Thus, the reference about cost estimation models developed in large environments, such as COCOMO, does not make sense for them. In fact the question of the applicability of the methods available nowadays in the local business firms always arises.

In section II we describe a framework in which the software experiments can be developed and interpreted. In section III we describe the two experiments carried out in the last years and section IV is the conclusion.

II. DESIGNING SOFTWARE EXPERIMENTS.

Some examples of software engineering experiments with students in which valuable knowledge has been obtained can be found in [1], [10] , [11]. Abdel-Hamid in a laboratory experiment found that people tend to make conservative estimates in software project control. It was a research on the empirical analysis of individual decision making behavior. Porter et al. in an experiment with forty-eight graduate students found that it is better to use systematic techniques to discover different, specific classes of faults, than to use general or checklist methods in software inspection activities. Finally, Scanlan demonstrated that structured flowcharts outperform pseudocode in helping programmers to understand algorithms, that was against the current assumption at that time.

An experimentation framework for developing experiments was defined by Basili et al. in [2], to help structure the experimental process and to provide a classification scheme for understanding and evaluating experimental studies. A schema of this framework can be seen in Table I. Other references for the designing experiments are [8] and [9]. The definition of an experiment has six components, namely motivation, object, purpose, perspective, domain, and range. During the definition phase, we develop our intuitive understanding of a high-level problem into a precise specification of an experiment that could contribute to its solution. The experiment is planned in detail in the second phase of the framework. The experiment is actually carried out during the third phase. The results are interpreted in several ways.

The design of experiments does not have many different characteristics whether they are carried out in industry or with students. From the point of view of the definition the issues that can be tackled are those that can be replicated across a set of groups, representing different projects. Motivation, object, purpose and perspective and domain, can be varied and there seems to be no limit on the issues that can be undertaken. Planning, in which refers to statistical analysis methods and metrics selection and definition can be also similar to those of industry. It is in the operation stage where more differences could appear from the point of view of preparation and execution. Project control and data collection could be more demanding in a university setting than in an organizational business structure already established.

Interpretation should be done with care in both environments. In case of academic environments, extrapolation should be done with care to other environments. In the case of industry, the results should be accompanied of the parameters that describe the environment in which they have been obtained, since in industry there are large variations in the structure of the software organization. So, the experiments exposed in the next section could be viewed within the framework here reproduced.

Table I. Framework for experimentation.

		I. Definition			
Motivation	*Object*	*Purpose*	*Perspective*	*Domain*	*Scope*
Understand	Product	Characterize	Developer	Software Team	Single Project
Assess	Process	Evaluate	Maintainer	Software Project	Multi-Project
Manage	Model	Predict	Project Manager		Replicated
Improve	Metric	Motivate	Corpor. Manag.		Project
Validate	Theory		Customer		Blocked
			User		subject-project
			Researcher		
		II. Planning			
Design		*Criteria*		*Measurement*	
Experimental design		Direct Criteria		Metric Selection and	
Statisticalanalysis		Indirect Criteria		Definition	
methods				Data collection Methodology	
		III. Operation			
Preparation		*Execution*		*Data analysis*	
Pilot study		Data collection		Preliminary analysis	
Participant training		Data validation		Formal analysis	
		IV. Interpretation			
Interpretation context			*Extrapolation*	*Impact*	
Statistical framework			Sample representativeness	Visibility	
Study purpose				Replication	
Field of research				Application	

III. TWO EXPERIMENTS IN SOFTWARE MEASUREMENT.

The curriculum at our Faculty has allowed us to develop a business software system project during the last year of the B.Sc. in Informatics. So during this fifth year of undergraduate education, groups of students develop an accounting information system. The introduction of the software metrics courses in the undergraduate curriculum allowed us to adjust both courses by applying the content of the software metrics course to the software project. The courses in software metrics lasted approximately forty hours. Training in the function point techniques was done according to the classical textbooks [6] , [12].

Thus, since 1992 to 1995 (three academic years) students began to record all data concerning to their software projects. The people involved were those of the last course of the B.Sc. The number of people involved was 120 in the academic course 92-93, 80 in the course 93-94 and 100 in the course 94-95. All people were divided into groups of 3 people, mainly. However, about a 25% of the data was spoilt, because of inconsistencies or doubts in the recordings. The number of projects that remained at the end of the experiments was 24 (plus 21 more projects for validation of several hypothesis) for the first experiment and 46 for the second.

The size of the projects range from small to medium in a 4GL, according to the sizes given in Table III. Most of the projects can not be classified as “toy” problems, since they involve a remarkable amount of real software complexity.

For the purpose of collecting data, sheets similar to those of Grady and Caswell [7] were designed. In the first experiment the formats of the sheets were changed from one year to another, in order to collect additional data. The reason for this is that the use of auxiliary sheets tended to introduce mistakes in the recordings. In the second experiment the sheets remained unaltered for two years. To recover data “a posteriori” is almost an impossible task.

The fact that all projects have been developed in the same environment (mostly 4GL) and domain (accounting information systems) allows the value of the "technical complexity adjustment" variable of the function points methods to be set to constant, and also allows us to examine the relationships among the variables.

Regarding the reliability of the effort data, it can be said that students did not lie, since they were assured that the hours recorded did not have any effect on the final grades obtained. No intent of cheating in the effort spent was observed. However, it was observed that lack of care in the recording of the data could be a major factor in the result of the study. Only under strict scrutiny of the teacher, information could be guaranteed to be reliable. An important factor for a correct recording of the data is the motivation of the students. Thus, with the purpose of being as reliable as possible, some groups recorded times in minutes, while others recorded rounding to half hours and in some cases whole hours. For the statistical analysis, all data was rounded in hours. The reason for the spoilt data was because of the lack of care in the recording of the data during the development of the projects. Also, it was not difficult to detect and correct some inconsistencies among the members of the groups. Since all hours reported were “net hours” a confident relationship between effort and size can be studied. This can make a difference with the data coming from industry, that is usually recorded in persons-month disregarding time rests or times that are not strictly devoted to the project. In this sense, comparison of effort obtained in different settings must be made with care.

III.1. Study of the function points method.

In this experiment the specific interest lied in examining the relationships between measures of the software product and the development effort. Given the dissatisfaction that exists with the lines of code -Loc- metric as a measure of size, and the great variability that can be incurred on the counting process, we intended to explore the different relationships that could exist among the measures of size and the effort. The four metrics of interest where: Albrecht function points, MarkII function points, Loc and Effort of each project. Table II defines this experiment within the framework of Basili et al.

Albrecht introduced a measure about the "functionality" that an application delivers to the user. The characteristic of that metric is that it can be computed early in the software life-cycle. Albrecht's measurement method is called Function Point

Analysis -FPA- [6], and the counting process consists of: a) classifying the user functions as inputs, outputs, inquiries, logical internal files and interface files, b) defining the complexity of each function, c) calculating the unadjusted function point value, d) assessing the general application characteristics and counting the final function point value.

Table II. Definition of the first experiment

Motivation	to understand the relationships among different metrics
Purpose	characterize and evaluate the relationships
Object	the development process and the specification measures
Perspective	from the perspective of the software manager
Domain	software project
Scope	replicated project

This method has been the focus of several criticisms, such as interrater reliability of the function counts and applicability to different development environments. The second type of criticisms has generated different variations to the original proposal such as SPQR/20, Mark II and Entity-Relationship function point.

Mark II Function Point Analysis -MkII- is the more radical scheme and was proposed by Symons [12] as a better function point technique after noting several shortcomings of FPA. His proposal consisted of considering a system composed of logical input/process/output transactions.

Thus, the questions that we asked ourselves for our environment were:

1) What is the relationship between the function point counts given by MkII and Albrecht methods?
2) What is the relationship between Loc and FPA?
3) What is the relationship between Loc and MarkII?
4) What is the relationship between Effort and FPA?
5) What is the relationship between Effort and MarkII?
6) What is the relationship between Loc and Effort?

For each one of these research questions a corresponding statistical hypothesis was formulated and tested with the data collected (Table III). The TCA in table III is the value of the "technical complexity adjustment" variable, used for the final calculation of the function point value. It takes into account the characteristics of the environment in which the system is developed.

Table III. Data for the first experiment.

a) Function Point Data

	No. of Projects	Mean FP	Std. Dev. Mean	Range	TCA (all projects)
FPA	23	169.73	87.12	65.5-466.4	13
MARK II	23	104.08	59.32	50.4-307.3	16

b) LOC Informix-4GL data

No. projects	Mean Total LOC	Std. Dev	Range
22	2329	1500	835-8006

c) Second Set of Data (for the analysis of effort)

No. projects	Mean LOC Informix	Std. Dev	Range	Mean Total hours-person	Std. Dev	Range
21	3911	1913	1185-8116	473	264	142-1154

The outcome of the analysis of these questions is as follows. For the first, second and third questions a notable correlation was found between the variables. However, the statistical results gave the two function point methods can not be used interchangeably. The Albrecht function point method could not be applied in a project, due to the deficiencies appeared in the definition of "logical file". The most striking result obtained was that function points were not able to predict effort (questions 4 and 5). This was against the common assumption that function points allows us to plan in the early stages of a software project. And Loc had a greater relationship with effort than function points.

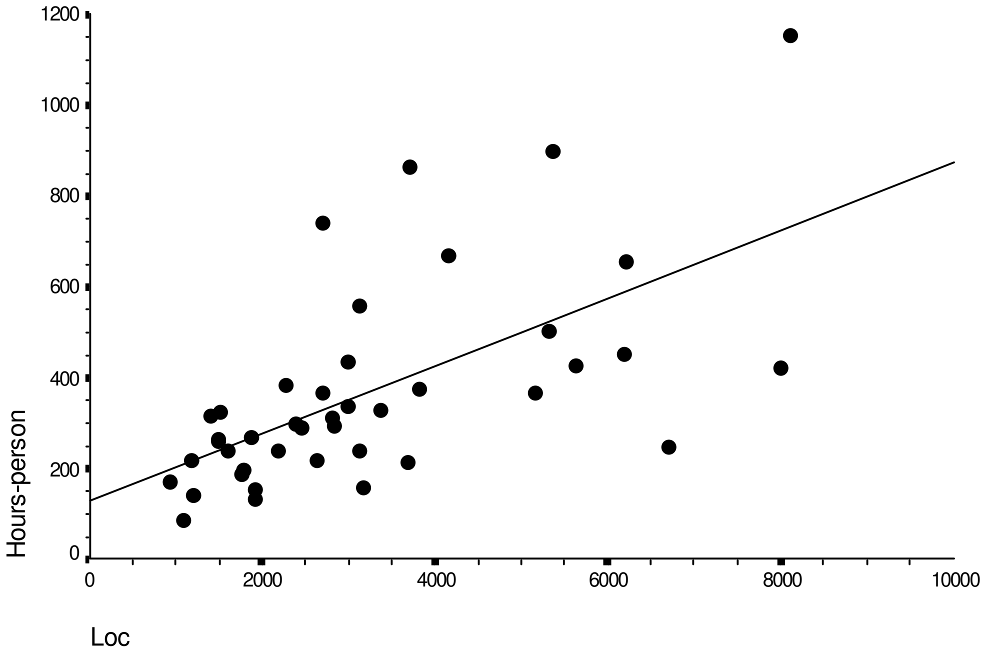

Figure 1. Effort versus Loc.

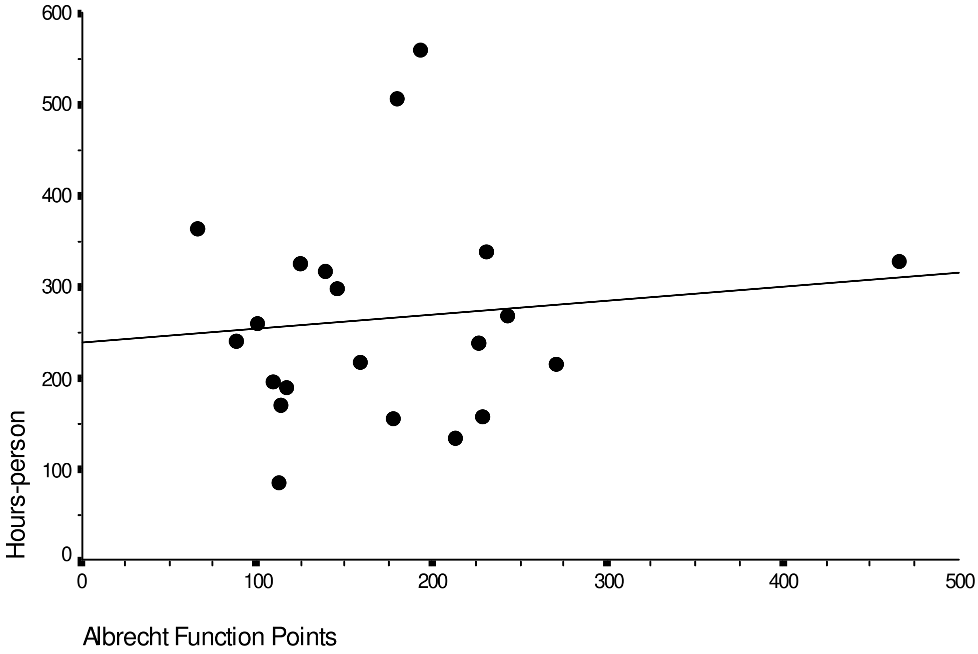

Figure 2. Effort versus Albrecht function points.

Figure 1 shows the plot of effort against Loc and figure 2 depicts the dispersion of effort against Albrecht function points.

How could we interpret the results in the light of all the data available from industry and academia? In fact the only data in which the reliability is less visible is that of the effort. We have to rely on the data reported by the students. But these doubts could also be extended to industry, where in some cases it is not clear how the effort has been accounted for. For the rest of the data, although making mistakes is unavoidable, no other sources of inaccuracies have been seen. This counterintuitive behavior can be explained by several causes: a) productivity and rate of learning are fluctuating among individuals and groups; b) no strict standards were followed so the human factor could have influenced more the results, and c) the supposed principles governing the effort-size relationship do not strictly apply to small sizes or to some environments. These results could arise in industrial environments if different projects allow such variations in productivity, or different companies accommodate different process capabilities.

III.2. Validation of a method for size estimation.

The main motivation for this experiment was to be conscious that estimation of software size is a crucial activity among the tasks of software management. Based on the estimation of the size of the software product, subsequent estimations of the effort required and work planning are made. Loc is a common measure of software size and is usually one of the independent variables in the equations for effort estimation.

There are several methods for estimating in the early stages the final loc of an application. A new method for estimating early in the software life-cycle the loc of a system was proposed by Verner and Tate [13]. It can be described as a “component-based method" -cbm-. The Verner-Tate's method estimates the size of the software by sizing individual components or modules first, and then obtains the system size by summing component sizes. The partition in components depends on the environment (type of software), and is not fixed. That approach tries to generalize the division in components that is done by function point analysis.

So, in this experiment the validation of the component-based method was tackled. This has been done through the analysis of 46 projects involving 117 000 loc of a fourth generation language. Table IV defines this experiment.

Table IV. Definition of the second experiment

Motivation	to improve the size estimation process
Purpose	evaluate an estimation method
Object	estimation process at early stages
Perspective	from the perspective of the software manager
Domain	software project
Scope	replicated project

In every project the corresponding systems and software engineering tasks, such as requirements specification, analysis, design, coding and documentation were carried out. The method of sizing by components was explained in detail according to the original article and data. 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 parts: menus, inputs and reports/inquiries. This division was found adequate and was done according to the tables provided by the article of [13].

The questions for this experiment were:

1) Is the cbm generally valid?

2) How does it compare with estimating Loc by the Mark II method?

The evaluation of the goodness is done according to the examination of the following parameters (see Conte et al. [3]):

- *Mean magnitude of relative error*, Mmre, is defined as $Mmre = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{e_i - \hat{e}_i}{e}\right|$, 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 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 Pred(0.1) = 0.9 means that 90% of the cases have estimates within the 10% range of their actual values. A standard criteria for considering a model as acceptable is Pred(0.25) $\geq$ 0.75. This means that at least 75% of the estimates are within the range of the 25% of the actual values.
- *Level of prediction*, Lpred, that is simply the quotient of the estimate divided by the actual value, Lpred = ê/e. Then we could compute the mean and standard deviation of Lpred for a set of cases.

Table VI shows the data coming from the evaluation of the cbm and Table V shows the evaluation of the MkII method for size estimation. For the Mark II method it can be observed that on average the level of prediction is 110% over the actual values and there is a standard deviation of 41%. The Mmre is on average 30% and the value for both Pred(0.25) and Mmre ≤ 0.25 is 0.53. This means that only 53% of the projects are within the range of the 75% of their actual values, and that 53% of the projects have an error less than 25%. In the light of these figures, it can not be said that the estimation of loc by unadjusted function points MkII performs well.

Comparing with the results for the cbm in table V, it can be seen that the cbm obtains better results. In table VI (using all data), the only condition in which the results are a little worse for the cbm is in Mmre ≤ 0.25, where on average inputs and reports/inquires have a 46% of cases and the MkII gives 53% for all the projects. Menus obtain much better performance compared to the MkII and inputs and reports/inquiries have better values in Pred(0.25) with a 57% on average satisfying it. But once some tuning of the regression has been made for the data in the cbm, eliminating extreme cases and outliers, the balance is clearly in favor of the cbm. Except for the Mmre where values remain similar or slightly better for two types, the cbm obtains much better values in the Lpred. Adding this to the fact that in one case we are estimating parts of an application and in the other case we estimate global loc, the inference is that estimating by components is a much better choice.

Table V. Evaluation of the MarkII method.

43 projects	Mean Value	Std. Dev	Min-Max values
Lpred	1.1	0.41	0.54-2.51
Mmre	0.3	0.29	0-1.51
PRED(0.25)	0.53		
Magnitude of RE ≤ 0.25	0.53		

The cbm has not presented here the impressive results that were published in the original article [13], but it has demonstrated that it has powerful capabilities as a size estimation method.

Table VI. Evaluation of the Component-Based Method

46 projects	Mean Value	Std. Dev (mean value)	Min-Max values	
Lpred				
Menus	1.04	0.08	1.00	1.36
Inputs	1.18	0.30	0.49	2.17
Reports/Inquiries	1.22	0.37	0.98	3.00
Mmre				
Menus	0.11	0.15	0.00	0.66
Inputs	0.40	0.38	0.00	1.48
Reports/Inquiries	0.40	0.42	0.00	2.28
Pred(0.25)				
Menus	0.88	0.26	0-1	
Inputs	0.57	0.31	0.1-1	
Reports/Inquiries	0.58	0.26	0-1	
Pred(0.25) ≥ 0.75				
Menus	0.82			
Inputs	0.30			
Reports/Inquiries	0.22			
Mmre ≤ 0.25				
Menus	0.84			
Inputs	0.46			
Reports/Inquiries	0.46			

III.3. Lessons learnt from the experiments.

The aim of the experiments was to show the students how these methods worked in practice. This would be a clear evidence to the students of the validity and extent of the software management methods. Direct consequences of the experiment are immediate availability of data and the motivation for the students to learn more about software management. Indirect consequences for both teachers and students is to understand a little more about how to manage people and groups. Also, it helps acquire experience for experimentation in industry.

The students formed some questions at the end of the projects when data was already becoming available and comparison with data from the bibliography was possible. It was peculiar that students tended to think that there results were wrong. The fact that the results were so varied among all groups tended to give confidence that things were correctly done. Data that is less subjective to the groups, such as the count of function points, was also cause of misunderstanding among the groups. However, this data, because of its calculability is less subjected to controversy.

It is clear that all students perform differently and that have different productivities, in spite of receiving the same education. The differences are mitigated when working in groups. Having a "baseline" student or project is similar to having a baseline project (and data) in industry. Nevertheless, the data coming from university setting could serve as reference for industrial studies.

Organizational guidelines would require some type of experiments projects lasting an academic year. All the results that could be expected should be planned for, even if they are not in agreement with results obtained by others. This helps to collect data that otherwise will not be available after the experiment is finished.

IV. CONCLUSIONS.

Several advantages are obtained in university environments: uniformity, more control (since there is no market pressure), cheap cost (except the additional hours of the teacher), motivation for learning software management. Projects of small to medium size can be carried out with success.

We have verified in our own setting some affirmations that are widely assumed about software measurement. The results allow us make a comparison with the data already available, and provide the teacher with the confidence needed to teach the software management techniques used nowadays. In some cases we have found some discrepancies with the original statements. Although that could simply mean that the environment is different, it could also signify that not all methods are generally valid.

Being critic, verifying and validating affirmations, is the only way to find the truths that could give us the confidence to teach the right approaches to software development and management. Those reflexive attitudes, if assumed by the students, could be the germ of future new ideas generated in the real software industry. Teachers are in charge of transmitting knowledge, but students can be a rich source of it.

V. REFERENCES.

[1] T. K. Abdel-Hamid, K. Sengupta and D. Ronan, "Software Project Control: An Experimental Investigation of Judgment with Fallible Information", *IEEE Trans. on Software Engineering*, , vol 29, No. 6, June 1993, pp 603-612

[2] V.R. Basili, R.W. Selby and D.H. Hutchens, "Experimentation in Software Engineering", ", *IEEE Trans. on Software Engineering*, Vol. 12, No. 7, July 1986, pp 733-743

[3] S.D. Conte, H.E. Dunsmore and V.Y. Shen, *Software Engineering Metrics and Models*, Benjamin/Cummings 1986

[4] J.J. Dolado, "A Study of the Relationships among Albrecht and Mark II Function Points, Lines of Code 4GL and Effort", *Journal of Systems and Software*, to appear.

[5] J.J. Dolado, "A Validation of the Component-Based Method for Software Size Estimation", submitted for publication

[6] J.B. Dreger, , *Function Point Analysis*, Prentice-Hall, 1989

[7] R.B. Grady and D.L. Caswell, *Software Metrics: Establishing a Company Wide Program*, Prentice-Hall, 1987

[8] B.A. Kitchenham, "Evaluating Softwar Engineering methods and Tools, Part 1: Evaluation context and evaluation methods", *ACM Software Engineering Notes*, vol. 21, no. 1, Jan. 1996; "Part 2: Selecting an appropriate evaluation method - technical criteria", vol. 21, no. 2, March 1996

[9] S.L. Pflegger, "Experimental Design and Analysis in Software Engineering, Part 1: the Language of Case Studies and Formal Experiments" vol.19. no.4, 1994, pp. 16-20; Part 2: How to Set Up and Experiment, Vol. 20, no. 1, pp 22-26; Part 3: Types of Experimental Design, vol. 20, no. 2, pp 14-16; Part 4: Choosing and Experimental Design, *ACM Sigsoft Software Engineering Notes*, vol. 20, no. 3, pp 13-15, July 1995, Part 5: Analyzing the Data, vol. 20, no. 5, pp 14-17, Dec. 1995

[10] A.A. Porter, L.G. Votta, Jr. and V. R. Basili, "Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment", *IEEE Trans. on Software Engineering*, vol 21, No. 6, June 1995, pp 563-575

[11] D.A. Scanlan, "Structured Flowcharts Outperform Pseudocode: An Experimental Comparison", *IEEE Software*, Sept. 1989, pp 28-36

[12] C.R. Symons, *Software Sizing and Estimating. MkII FPA (Function Point Analysis)*, John Wiley & Sons, 1991

[13] J. Verner and G. Tate, "A Software Size Model", *IEEE Trans. on Software Engineering* 18, No. 4 April 265-278 1992