Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 2004 Proceedings

Americas Conference on Information Systems (AMCIS)

December 2004

Assessing Productivity of UML-based Systems Analysis and Design: A DEA Approach

Qing Cao
University of Missouri-Kansas City

Assessing Productivity of UML-based Systems Analysis and Design: A DEA Approach

Qing Cao
University of Missouri-Kansas City
caoq@umkc.edu

ABSTRACT

Due to the recent economic woes, information systems (IS) departments are under enormous pressure to cut costs while maintaining productivity in software development projects. Improving software project productivity has become a critical issue for every organization especially under the current economy uncertainty. Previous studies focus mainly on the efficiency of the traditional SAD process and there have been no attempts, to our best knowledge, to explore the efficiency of OOSAD process. This study uses UML complexity metrics to measure the productivity of OOSAD projects. DEA models are developed to explore the efficiency of these projects. A preliminary empirical study is conducted to apply the proposed methodology. Finally, sensitivity analysis of DEA is used in the study to identify factors and ways to improve the project efficiency and managerial implications are also discussed.

KEYWORDS

Software engineering, Project Productivity, OOSAD, UML, DEA

INTRODUCTION

Due to the recent economic woes, information systems (IS) departments are under enormous pressure to cut costs while maintaining productivity in software development projects. However, software projects are infamous for being over budget, delivered late, and failing to meet expectations. According to the latest Standish Group CHAOS report (2003), project failure rates are as high as 65% of 13,522 information technology (IT) projects surveyed with estimated a 43% average overrun cost, a 82% average time overrun, and a 51% average coverage of required features and functions. Moreover, the lost dollar value for US projects in 2002 is estimated at $55 billion against $255 billion in overall project spending. As a result, improving software project productivity has become a critical issue for every organization especially under the current economy uncertainty. One way to increase the productivity of software project is to improve the performance of systems analysis and design (SAD) process (Mahmood et al., 1996). The aim of this research is two-fold. First, we find and utilize the proper productivity measurement for the SAD process. Second, we propose a methodology to examine the efficiency of the SAD process in software development.

The most commonly used SAD productivity measurements are function points (FPs), lines of codes (LOCs), or a combination of FPs and LOCs (Banker and Slaughter, 1997). However, because of the increasing popularity of object-oriented (OO) technology in the SAD environments, traditional productivity metrics such as FPs and LOCs are no longer sufficient for characterizing, assessing, and predicting the performance of object-oriented systems analysis and design (OOSAD) process (Basili et al., 1996). According to Siau and Cao (2001), UML is not only the de facto modeling language standard for specifying, visualizing, constructing, and documenting the components of software systems, but it has also been accepted by the Object Management Group (OMG) as a standard language for object-oriented analysis and design. This research uses the UML complexity metrics developed by Rossi and Brinkkemper (1996) and calculated by Siau and Cao (2001) to measure the productivity of UML-based OOSAD projects.

Previous studies focused mainly on the efficiency of the traditional SAD process. To our best knowledge, there have been no attempts to date that explore the efficiency of OOSAD process. Data envelopment analysis (DEA) is a widely accepted benchmarking approach in exploring productivity efficiency (Charnes et al., 1981; Banker et al., 1991; Joro et al., 1998) and thus it is the methodology of choice for this study.

The contributions of this research include: 1) it uses UML complexity metrics to measure the productivity of the UML-bases OOSAD projects; 2) it creates DEA models to explore the efficiency of UML projects; 3) it applies the proposed methodology to a preliminary empirical study based on study OOSAD projects; 4) sensitivity analysis of DEA is also used in the study to identify factors and ways to improve the project efficiency.

LITERATURE REVIEW

UNIFIED MODELING LANGUAGE (UML)

UML has become an industry standard for the presentation of various design artifacts in object-oriented software development (Selonen et al., 2003; Siau and Cao, 2001). UML is a modeling language that is easily learned by system developers and an industry standard, which supports communication between the various software project stakeholders. The increased popularity of UML provides a real opportunity for formal OOSAD methods to be used on a daily basis within the systems development lifecycle (Henderson-Sellers, 2002). UML is used extensively by OO systems developers and as such is important in the analysis and design process for many organizations. UML consists of nine separate diagrams which can be divided into two categories – structural and behavioral diagrams.

Structural diagrams

Class diagrams, object diagrams, component diagrams, and deployment diagrams comprise the static models of UML (Booch, G., Rumbaugh, J., and Jacobson, I. 1999). Static models represent snapshots of the system at a given point or points in time, and do not relate information about how the system achieved the condition or state that it is in at each snapshot.

Behavioral diagrams

Use case diagrams, sequence diagrams, collaboration diagrams, activity diagrams, and statechart diagrams make up the dynamic models of UML. In contrast to the static diagrams, the dynamic diagrams in UML are intended to depict the behavior of the system as it transitions between states, interacts with users or internal classes and objects, and moves through the various activities that it is designed to accomplish (Booch, G., Rumbaugh, J., and Jacobson, I. 1999).

PRODUCTIVITY METRICS

Researchers and managers have attempted to measure the productivity of software development projects by means of several related but different approaches, first by counting the lines of code in the finished product, and second by counting the number of function points in the software. This research uses an alternative set of metrics, developed by Rossi and Brinkkemper (1996) to quantify the complexity of the modeling methods which underlie the software itself.

LOC (Lines of Code)

Software metrics in general attempt to relate the size, structure, quality, and reliability of software to its complexity, cost, adoption, or success (Fenton and Pfleeger, 1997). As such the basic ideas and premise of the software metrics approach appear to support the purposes of this study. However, since this study purports to examine the productivity of the UML modeling method, the methods typically used in assessing software productivity do not really apply here. For example, one of the primary means of assessing software complexity as a way of quantifying development efficiency as related to costs is to measure the number of lines of code in the program under investigation, and (typically) compare that with the effort in time expended to create the software. While LOC does not directly measure program development efficiency, it does provide an internal measure of how large the program is, which can be considered as only one of many development process indicators (Albrecht, and Gaffney, 1983). To complicate matters, Halstead (1977) found that there were at least six different ways to count lines of code.

Function Points

Since LOC does not appear to provide an entirely reasonable or satisfactory estimate of program/system development efficiency, Function-Point (FP) analysis was developed as an alternative. Albrecht (1979) examines a program or system from the perspective of the number of functions or modules it incorporates. A number of serious issues have been identified with the function point approach.

These problems include a subjectivity issue – regards personal judgments involving initial values, double counting functions – whether the same function can be accessed by multiple modules), counter-intuitive values – the formulas can produce results that do not make sense in some instances, accuracy, and using FP too early in the development cycle – FP is intended

to be used later in the cycle (does not react well to requirement changes), and not early (when it would be most beneficial) (Fenton and Pfleeger, 1997). Many other problems with FP have also been identified, thus rendering its utility questionable.

Complexity

Software complexity is typically assessed using such measures as McCabe's Cyclomatic number, Haltstead's Software Science model, and Oviedo's data flow complexity model (Fenton and Pfleeger, 1997). In 1988 Weyuker proposed a number of evaluations for software complexity measures. She ended by proposing that none of the complexity measures in existence at the time adequately captured what is meant by complexity, and called for more research on the subject.

Modeling Method Complexity Metrics

Rossi and Brinkkemper's (1996) research proposed and developed a relatively easy to use and straightforward means to quantitatively measure system development methods. Specifically, the metrics are based on metamodel techniques, and purport to measure the complexity of the method under analysis. According to Rossi and Brinkkemper (1996), complexity is critical to measure because researchers supposed complexity to be closely related to how easy a specific method is to use, and also how easy the method is to learn. One of Rossi and Brinkkemper's more crucial caveats is that the measured complexity of a given system does not solely translate into less complex methods being superior to more complex methods. A set of seventeen complexity metrics are proposed by Rossi and Brinkkemper (1996) and it includes 9 independent metrics, 3 aggregate metrics, and 5 method-level metrics.

UML Complexity Metrics

Siau and Cao's (2001) research applied Rossi and Brinkkemper's complexity metrics to the Unified Modeling Language (UML). Siau and Cao (2001) also compared UML's complexity with 36 OO techniques from 14 methods, as well as each of the 14 methods in aggregate. One of Siau and Cao's (2001) noteworthy findings is that UML is far more complex (from 2 to 11 times more complex) in aggregate than any of the other 13 methods. The size relative overall complexity highlights one of the issues regarding UML, that it can appear overwhelming to those new to UML. This can affect how people use the modeling language, and therefore also affect the efficiency of the Additionally, when human cognitive limitations to short term memory are added to this mix, UML can appear even more difficult to master. More importantly, Siau and Cao (2001) computed UML complexity based on the 17 complexity metrics proposed by Rossi and Brinkkemper (1996). In this research, UML complexity is used as the productivity measurement for OOSAD projects.

METHODOLOGY

INPUT VARIABLE

Prior research (Chatzoglou and Soteriou, 1999; Banker and Slaughter, 1997; Mahmood et al., 1996; Banker et al., 1991) shows that labor hours are the most commonly used measure of the software project development effort for a software project. For instance, Banker et al. (1991) used total labor hours as the single input measure for investigating the productivity of software maintenance projects. Mahmood et al. (1996) used multi-dimensional inputs involving labor hours spent on different stages of SAD process. This study uses total project labor hours as the single input variable due to the exploratory nature of the research.

Input variable: total project labor hours

OUTPUT VARIABLES

Since the goal of this research is to assess the productivity efficiency in an OOSAD environment, UML complexity metrics are used as the output measurements of UML projects. In order to maintain consistency of the metrics, we only use the independent metrics of UML complexity. Table 1 shows nine independent metrics for UML adopted from Siau and Cao (2001).

UML Diagram/Metrics	n (O_T)	n (R_T)	n (P_T)	Po (M_T, o)	$\overline{P}_o$ (M_T)	P_R (M_T, e)	$\overline{P}_R$ (M_T)	Ro (M_T, o)	$\overline{R}_o$ (M_T)
Class	7	18	18	12	1.71	22	1.22	18	2.57
Use Case	6	6	6	6	1	5	0.83	6	1
Activity	8	5	6	6	0.75	1	0.2	5	0.63
Sequence	6	1	5	4	0.67	6	6	1	0.17
Collaboration	4	1	7	4	1	8	8	1	0.25
Object	3	1	5	5	1.67	3	3	1	0.33
StateChart	10	4	11	10	1	2	0.5	4	0.40
Component	8	10	9	8	1	13	3.6	10	1.25
Deployment	5	7	5	5	1	8	1.14	7	1.40

Table 1: Independent Metrics for UML

Legend:
n (O_T) – count of object types per technique
n (R_T) – count of relationship types per technique
n (P_T) – count of property types per technique
Po (M_T, o) – count of number of properties for a given object type
$\overline{P}_o$ (M_T) – average number of properties for a given object type
P_R (M_T, e) – number of properties of a relationship type and its accompanying role types
$\overline{P}_R$ (M_T) – average number of properties per relationship type
Ro (M_T, o) – number of relationship types that can be connected to a certain object type
$\overline{R}_o$ (M_T) – number of relationship types that can be connected to a certain object type

Output variables: n (O_T), n (R_T), n (P_T), Po (M_T, o), $\overline{P}_o$ (M_T), P_R (M_T, e), $\overline{P}_R$ (M_T), $\overline{R}_o$ (M_T)

A brief illustration of how to computer an output variable n (O_T)

Assume an OOSAD project renders one for each of the nine UML diagrams.

We can get n (O_T) as following:

n (O_T) = 7*1+6*1+8*1+6*1+4*1+3*1+10*1+8*1+5*1 = 57

By the same taken, we can derive other output variables using the same technique.

DATA ENVELOPMENT ANALYSIS

In the present research, we use Data Envelopment Analysis (DEA), a non-parametric linear programming method, to assess the project productivity. DEA was introduced in 1978 by Charnes, Cooper, and Rhode and it is a fractional programming model that estimates the relative efficiencies of a homogeneous set of units by considering multiple sets of inputs and outputs. The efficiency is obtained by computing the ratio of the weighted sum of outputs to the weighted sum of inputs. DEA has been utilized extensively for comparing the relative efficiencies of DMUs such as schools, hospitals, bank branches, and other environments Banker et al. (1997).

$$\text{Max} \quad \frac{\sum_{k=1}^{s} v_k y_{kp}}{\sum_{j=1}^{m} u_j x_{jp}} \qquad (1)$$

$$\text{s.t.} \quad \frac{\sum_{k=1}^{s} v_k y_{ki}}{\sum_{j=1}^{m} u_j x_{ji}} \leq 1 \text{ for } i = 1 \ldots p \ldots n$$

$$v_k,\ u_j \geq 0 \text{ for } k = 1 \ldots s, \text{ and } j = 1 \ldots m$$

The relative efficiency of a DMU *p* is obtained by solving the following fractional programming model proposed by Charnes et al. (1981).

where *p* is the DMU being analyzed, *k* represents the number of outputs, *j* represents the number of inputs, *i* is the number of decision making units, y_{ki} is the amount of output *k* produced by DMU *i*, x_{ji} is the amount of input *j* used by DMU *i*, v_k is the weight given to output *k*, and u_j is the weight given to input *j*.

Formulation (1), which is referred to as the ratio model, can easily be converted to the linear programming (LP) model as shown in formulation (2):

The LP model in formulation (2) is solved *n* times in order to determine the relative efficiencies of all the DMUs. The model allows each DMU to effectively select optimal weights that maximize its output to input ratio, but at the same time the constraints prevent the efficiencies of the *n* DMUs computed with these weights from exceeding the value of 1. A relative efficiency score of 1 indicates that the DMU under consideration is ratio efficient, whereas a score less than 1 indicates that it is ratio inefficient.

$$\text{Max} \quad \sum_{k=1}^{s} v_k y_{kp} \qquad (2)$$

$$\text{s.t.} \quad \sum_{j=1}^{m} u_j x_{jp} = 1$$

$$\sum_{k=1}^{s} v_k y_{ki} - \sum_{j=1}^{m} u_j x_{ji} \leq 0 \text{ for } i = 1 \ldots p \ldots n$$

$$v_k,\ u_j \geq 0 \text{ for } k = 1 \ldots s, \text{ and } j = 1 \ldots m$$

Benefits of DEA in Assessing Project Productivity

According to Charnes et al. (1981), DEA has the following advantages in assessing project productivity:

- It does not require functional relationships between inputs and outputs.
- Multiple inputs and outputs can be considered concurrently.
- It has the ability to identify inefficient projects.
- Using DEA sensitivity analysis, the sources and amounts of inefficiency for each inefficient project can be found.

There are numerous DEA software packages available in the market. In this research, we used DEA Excel Solver coded by Joe Zhu (2003).

A PRELIMINARY EMPIRICAL STUDY

To ensure applicability of the proposed OOSAD productivity efficiency model, an extensive empirical validation process is needed. In this section we describe a preliminary empirical study that was conducted based on UML projects from an MBA level OOSAD class during a semester at a state university in the Midwest.

DATA COLLECTION

MBA students who participated in the projects were required to be part of a team of two people and 15 teams were established in this research. The demographic breakdown of the class during the semester was 60% computer engineers, 30% business analysts, and the rest 10% full MBA students major or minor in MIS. Rather than enforcing a particular topic, each team was given the flexibility to choose an OOSAD project by itself. Having identified an appropriate project, each team went through two stages (analysis and design) of the software development life cycle (SDLC) using all nine diagrams of UML at least one time. Each team submitted a report at the completion of the project and time spent on the project was recorded. All of the fifteen OOSAD projects were used in the study. Since all the projects are developed at the same time (same semester) using the same technology (UML), project inefficiency cannot be attributed to such factors as different design tools. Table 2 represents values of nine output variables (independent metrics) and one input for the 15 OOSAD projects.

Projects/Metrics	**Input**	**Outputs**								
	Time (hours)	n (O_T)	n (R_T)	n (P_T)	Po (M_T, o)	$\overline{P}_o$ (M_T)	P_R (M_T, e)	$\overline{P}_R$ (M_T)	Ro (M_T, o)	$\overline{R}_o$ (M_T)
Project 1	47	71	90	126	86	13.8	82	42.78	70	10.88
Project 2	36	95	76	86	88	15.14	89	35.55	72	11.15
Project 3	34	93	98	95	97	13.47	126	33.14	65	9
Project 4	31	75	70	126	87	13.51	90	26.66	89	13.54
Project 5	42	94	91	83	90	16.6	84	42.15	87	14.18
Project 6	48	82	84	89	94	13.18	100	47.84	60	14.35
Project 7	47	111	90	117	80	15.93	85	44.64	83	10.68
Project 8	45	88	90	107	81	10.8	96	35.41	73	11.21
Project 9	36	81	77	100	106	15.89	69	37.73	73	10.55
Project 10	47	100	91	112	86	17.85	130	30.26	75	11.22
Project 11	46	77	83	121	83	12.55	116	26.54	101	11.62
Project 12	36	104	98	90	75	12.47	114	38.81	62	13.43
Project 13	48	81	92	104	99	15.47	118	39.48	100	11.45
Project 14	44	87	85	103	91	15.47	127	26.96	68	8.58
Project 15	49	84	60	107	94	14.18	113	44.55	89	13.58

Table 2: Input and Output Variables

RESULTS AND DISCUSSION

DEA was applied in two stages of data analyses. First, DEA was used to find relative efficient projects. Relative efficiency of any project is attainted only when:

a) None of its outputs can be increased without either (i) increasing one or more of its inputs, or (ii) decreasing some of its outputs;
b) None of its inputs can be decreased without either (i) decreasing some of its outputs, or (ii) increasing some of its other inputs.

Second, a sensitivity analysis of DEA was performed to seek the causes of the inefficient projects. By and large, inefficiencies are caused by input and/or output slacks. An input slack for a project means that it can reduce its input by the

slack amount without reducing the output(s) while an output slack means that it will have to increase its output (s) by the slack amount to become efficient (Mahmood et al., 1996).

OOSAD PROJECT EFFICIENCY

In this research, DEA rendered the relative efficiency for 15 OOSAD projects using nine complexity metrics of UML (independent metrics). All the projects were considered as efficient if their relative efficiency ratios equaled 1. On the other hand, projects were regarded as inefficient if they obtained a relative efficiency ratio less than 1. Table 3 shows a summary of the project efficiency in this research. The results indicate that 5 OOSAD projects (33.34%) were deemed as efficient while 10 OOSAD projects (66.66%) were rated as inefficient.

Efficiency	Number of Projects	Percentage
1.0000	5	33.34
<1.0000	10	66.66
Total	15	100

Table 3: Summary of Project Efficiency

SENSITIVITY ANALYSIS

As mentioned previously, sensitivity analysis of DEA helps to find the causes of inefficiencies and as such it provides insightful information of how to increase efficiency (productivity) of a project. Table 4 represents input slacks (Excess Input) of the fifteen projects. Column 2 of table 4 shows the efficiency ratio in a descending order along with original input used in each project (column 3), while column 4 shows the input slacks. The results illustrate that projects 2, 3, 4, 9, and 12 were efficient, and all the other projects were inefficient. Among the inefficient projects, the efficiency ratio ranks from 0.7838 to 0.9838. The results also indicate that the efficient projects used less time (34.6 hours on average) than those of inefficient projects (46.3 hours on average). Column 4 of table 4 shows from an input standpoint how and to what degree the inefficient project teams can make their projects efficient. For instance, with an efficiency ratio of 0.7838 (less than 1), project 8 is an inefficient project. If they could reduce the overall project time from 45 hrs to 35 hrs, project 8 would become efficient without changing current output values.

		Input	Input Slacks
DMU Name	Efficiency	Time (hours)	Time (hours)
Project 4	1	31	0
Project 3	1	34	0
Project 2	1	36	0
Project 9	1	36	0
Project 12	1	36	0
Project 5	0.9838	42	1
Project 6	0.9255	48	4
Project 7	0.9077	47	4
Project 1	0.8955	47	5
Project 10	0.8916	47	5
Project 15	0.8853	49	6
Project 13	0.8645	48	7
Project 14	0.8531	44	6
Project 11	0.8216	46	8
Project 8	0.7838	45	10

Table 4: Slack (Excess Input)

Table 5 shows the efficiency ratio in descending order (column 2) and output deficiencies were shown between column 3 and column 11. Table 5 demonstrates, in contrast to table 4, from an output view how and to what degree the inefficient project teams can make their projects efficient. Take project 8 again for example, in order to achieve efficiency, project 8 needs to enhance its outputs – 6 out of 9 in this case. More specifically, the project 8 team needs to increase various outputs, i.e., n (O_T) – count of object types per technique by 7.8, Po (M_T, o) – count of number of properties for a given object type by 1.8, $\overline{P}_o$ (M_T) – average number of properties for a given object type by 2.6, P_R (M_T, e) – number of properties of a relationship type and its accompanying role types by 11.5, Ro (M_T, o) – number of relationship types that can be connected to a certain object type by 1.7, and $\overline{R}_o$ (M_T) – number of relationship types that can be connected to a certain object type by 2.6.

	Output Slacks									
Projects/Metrics	Efficiency	n (O_T)	n (R_T)	n (P_T)	Po (M_T, o)	$\overline{P}_o$ (M_T)	P_R (M_T, e)	$\overline{P}_R$ (M_T)	Ro (M_T, o)	$\overline{R}_o$ (M_T)
Project 4	1	0	0	0	0	0	0	0	0	0
Project 3	1	0	0	0	0	0	0	0	0	0
Project 2	1	0	0	0	0	0	0	0	0	0
Project 9	1	0	0	0	0	0	0	0	0	0
Project 12	1	0	0	0	0	0	0	0	0	0
Project 5	0.9838	7.7	5.1	38.9	20.6	0.5	17.5	0	0	0
Project 6	0.9255	45.3	35.9	22.5	0	2.4	38.6	0	17	2.1
Project 7	0.9077	0	14.8	0.4	25.7	0.7	28.7	0	0	4.2
Project 1	0.8955	24.9	0.9	0	36.9	4.7	5.6	0	21.3	2.5
Project 10	0.8916	4.7	10.1	45.1	32.1	0	0	7	35.3	5.3
Project 15	0.8853	26.3	44.1	18.4	17.3	3.2	0	0	0	1.7
Project 13	0.8645	24.2	6.8	37.8	9.3	1.5	0	0	0	5
Project 14	0.8531	10.9	13.8	21.1	15.4	0	0	7.9	18.2	4
Project 11	0.8216	16.9	7.2	22.7	23.4	3.6	0	6.9	0	3.5
Project 8	0.7838	7.8	0	0	1.8	2.6	11.5	0	1.7	2.6

Table 5: Slack (Deficient Outputs)

CONCLUSIONS AND LIMITATIONS

In this research, we present a framework to assess the efficiency of OOSAD projects using independent metrics of the UML complexity metrics calculated by Siau and Cao (2001). To our best knowledge, similar research aimed at exploring the efficiency issues of OOSAD process has not been performed to date,. Data Envelopment Analysis models were developed to implement the proposed framework. Results from a preliminary empirical study based on classroom OOSAD projects indicate that the framework and DEA used in this study can be viable not only to detect inefficiencies of OOSAD projects, but also to provide insightful information and guidance for project managers in terms of how to improve of the OOSAD projects.

However, this research has several limitations. First, we used OOSAD projects data from a classroom setting (exploratory in nature), and thus generalization of the findings might be less convincing. We are currently gathering project data from software companies and will apply the proposed framework to the new data. Second, we used only one set of the UML complexity measures, namely, independent metrics. In the future, we will expand the output variable to include aggregate metrics and method-level metrics. Finally, a single input variable (total labor hours) was used in this research. In the future, we will use multi-dimensional inputs.

REFERENCES

1. Albrecht, A., 1979, "Measuring Application Development Productivity", Proc. IBM Develop. Symposium, Monterey, CA, Oct. 14-17, pp 83.
2. Albrecht, A. J. & Gaffney, J. E., Jr. 1983. Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. IEEE Transactions on Software Engineering, SE 9(6): 639-648.
3. Charnes, A., Cooper, W. W., & Rhodes, E. 1981. Evaluating Program and Managerial Efficiency: An Application of Data Envelopment Analysis to Program Follow Through. Management Science, 27(6): 668-697.
4. Chatzoglou, P. D. & Soteriou, A. C. 1999. A DEA framework to assess the efficiency of the software requirements capture and analysis process. Decision Sciences, 30(2): 503-531.
5. Banker, R. D. D., Srikant M.; Kemerer, Chris F. 1991. A Model to Evaluate Variables Impacting the Productivity of software maintenance projects. Management Science, 37(1): 1-18.
6. Banker, R. D. & Slaughter, S. A. 1997. A field study of scale economies in software maintenance. Management Science, 43(12): 1709-1725.
7. Basili, V. R., Briand, L.C., Melo, W. L. 1996. IEEE Transactions on Software Engineering. 22(10): 751 -761.
8. Booch, G., Rumbaugh, J., and Jacobson, I. 1999, *The Unified Modeling Language User Guide*, MA: Addison-Wesley.
9. Fenton, N. and Pfleeger, S., 1997, *Software Metrics A Rigorous and Practical Approach*, PWS Publishing, pp. 243-278.
10. Halstead, M., 1977, *Elements of Software Science*, Elsevier, New York.
11. Henderson-Sellers, B. 2002. The use of subtypes and stereotypes in the UML model. Journal of Database Management, 13(2): 43-50.
12. Joro, T., Korhonen, P., & Wallenius, J. 1998. Structural comparison of data envelopment analysis and multiple objective linear programming. Management Science, 44(7): 962-970.
13. Mahmood, M. A., Pettingell, K. J., & Shaskevich, A. I. 1996. Measuring productivity of software projects: A data envelopment analysis approach. Decision Sciences, 27(1): 57-80.
14. Rossi, M. and Brinkkemper, S., 1996. "Complexity Metrics for Systems Development Methods and Techniques," Information Systems, Vol. 21, No. 2 pp. 209-227.
15. Siau, K. & Cao, Q. 2001. Unified Modeling Language (UML) - a complexity analysis. Journal of Database Management, 12(1): 26-34.
16. Standish Group CHAOS Report (2003) – http://www.standishgroup.com/sample_research/index.php
17. Selonen, P., Koskimies, K., & Sakkinen, M. 2003. Transformations between UML diagrams. Journal of Database Management, 14(3): 37-55.
18. Weyuker, E. J. 1988. Evaluating Software Complexity Measures. IEEE Transactions on Software Engineering, 14(9): 1357-1365.
19. Zhu, J., 2003, *Quantitative Models for Performance Evaluation and Benchmarking: DEA with Spreadsheets and DEA Excel Solver*, Kluwer Academic Publishers, Boston.