Analysis of Effort Estimation based on Software Project Models

Pichai Jodpimai, Peraphon Sophatsathit, and Chidchanok Lursinsap
Advanced Virtual and Intelligent Computing (AVIC) Center
Department of Mathematics, Faculty of Science
Chulalongkorn University
Bangkok, Thailand
E-mail: pichai.j@student.chula.ac.th, peraphon.s@chula.ac.th, lchidcha@pioneer.netserv.chula.ac.th

Abstract—**This paper investigates the interrelationship among various measured characteristics of a software project, ranging from project model, size, and metrics used to govern the administration of the project. By analyzing various dimensions of project characteristics based on the underlying model, metrics and project technicality such as language and development paradigm, our findings reveal that certain metrics and models are not suitable for small project since they possess insufficient information to extract and analyze the inherent characteristics of the project. As such, project managers should pay attention to proper selection of project parameters that are conducive toward accurate estimations.**

I. INTRODUCTION

Software development effort estimation is the process of forecasting the software effort to estimate software costs of both development and maintenance. Such estimates may be used as inputs to analyze project investment. Software researchers and practitioners have provided effort estimation for several decades. Most of them focused on the construction of formal software effort estimation models, such as Putnam's SLIM, COOMO 81, COOMO II, COCOTS, Kemerer and Albrecht-Gaffney, that can be applied to measure on different project scales (small, medium, and large). We employed five software projects ranging from small (less than 10,000 SLOC) to medium (10,000 – 100,000 SLOC) scale for estimating software effort by means of the aforementioned estimation models. Conventional and object-oriented metrics were used to determine the relationship among metrics, efforts, and project size.

The organization of this paper is as follows. Section II discusses some literature reviews on software project models and software metrics. Section III elucidates the analysis of this study. Section IV explains the experimental results. Conclusion and potential future work are discussed in the last section.

II. RELATED WORK

This section provides some background information on software project models and software metrics to be used in this research work.

A. Software Project Models

For years, many software cost estimation techniques and their underlying models have been developed uninterruptedly, some of which are shown in Table I and described below.

TABLE I
SOME COST ESTIMATION TECHNIQUES

Year	Proposed by	Objectives
2003	Yuan Zhao, et al.	explored the use of ER model for the estimation of software cost, and built a multiple regression model for software cost estimation.
2006	Issa A., et al.	tried to compensate for the unavailability of software cost estimation models that fit different software development environments in the early stages of the software development life cycle.
2007	Li, Y.F., et al.	introduced the selection of appropriate project subsets (project selection) by genetic algorithm.
2008	Keung, J.W.	proposed a novel method for the determination of the theoretical maximum prediction accuracy (TMPA) in the application of analogy-based software cost estimation. Specifically, they determine TMPA of analogy using a unique dynamic KNN approach to simulate and optimize the prediction system.

COnstructive COst MOdel (COCOMO) [1][2] is used to estimate software cost. It was first published in 1981 (COCOMO 81) and in 1997 (COCOMO II). Some differences between COCOMO 81 and COCOMO II are as follows: COCOMO 81 has 63 data points, uses Kilo Deliverable Source Instructions (KDSI) to measure the project size and three development modes to be represented by scale factors. In contrast, COCOMO II has 161 data points, uses KSLOC for project size, and five scale factors.

SLIM model [3] is an empirical software effort estimation model developed by Dr. Lawrence Putnam. It describes time and effort required to finish a software project of specified size. SLIM (Software Life cycle Management) is the name given by Putnam as the tool for the model.

Walston-Felix model [4], developed by C.E. Walston and C.P. Felix in 1977, is a method of programming measurement and estimation.
Bailey-Basili model [5] is based on data collected by organization which captures its environmental factors and the differences among given projects.
Doty model, published in 1977, is used to estimate efforts for Kilo lines of code (KLOC).
Albrecht-Gaffney model [6], established by IBM DP Services Organization, uses function point to estimate efforts.
Kemerer model [7] is a cost estimation model using function points and linear regression.
Matson, Barrett and Mellichamp model [8] develop a software cost estimation model using function points.

B. Classification of software metrics

Software metric is a measure of some properties of a program to estimate size and effort, improve software quality, and reduce further maintenance needs. Many software metric works have been developed continuously by researchers and practitioners for educational and commercial purposes as shown in Table II. They are categorized in two groups, namely, conventional and object-oriented metrics.

TABLE II
PREVIOUS WORKS ON SOFTWARE METRIC THEORIES

Year	Proposed by	Objectives
1993	Bush, M. and Ashley, N.	Metric Education Toolkit (Metkit) responded to the need for more material with which to teach measurement techniques as they apply to software engineering.
1994	Chidamber and Kemerer	introduced a metrics suite for object-oriented design based on Inheritance Tree.
1996	Basili, V., et al.	presented the results of a case study in which an incremental approach was used to better understand the effort distribution of releases and build a predictive effort model for software maintenance releases.
1998	Sen-Tarng Lai and Chein-Chiao Yang	introduced a two-layer metric combination (TLMC) model, which was based on a dynamically weighted linear combination for individual software metric combinations.
2001	Hastings and Sajeev	proposed a Vector Size Measure (VSM) to measure the size of software systems and to classify software systems, and introduced a Vector Prediction Model (VPM) to estimate development effort early in the software life cycle.
2003	Washizaki, et al.	presented a new metric method based on reusable components, which was used for measuring comprehensibility and reusability of OO components.
2005	Aine Mitchell and James F. Power	proposed a new metrics to quantify coupling at different layers of granularity such as class-class and object-class level.
2006	W. Eric Wong, et al.	proposed a statistical hypothesis testing methodology to verify "retrospectively."
2007	Da Deng, et al.	presented an empirical study on software effort estimation using an integrated data mining approach in which a series of procedures such as data visualization, feature selection, and modeling were conducted.
2008	Mingzhi Mao and Yunfei Jiang	organize the set of OO metrics in a coherent framework, identified relationships between existing object-oriented metrics and software quality metrics, and provided software quality metrics in a form of mathematical formulas.
2008	Rudiger Lincke, et al.	showed that existing software metric tools interpret and implement the definitions of object-oriented software metrics differently.

1) conventional metrics

Source Lines of Code (SLOC) is the number of Lines of Code in a software project that does not count blank lines, comment lines, and library [9]. It is popularly used for estimating software cost in COCOMO II.
Deliverable Source Instructions (DSI) is similar to SLOC. The difference between DSI and SLOC is that "if-then-else" statement, for example, would be counted as one SLOC, but might be counted as several DSI.
Function Points (FP) [10], defined by Allan Albrecht at IBM in 1979, is a unit of measurement to express the amount of software functionality. Function point analysis (FPA) is the method of measuring the size of software. It disregards language, technology, development method, and platform, but considers data function (internal logical files, external interface files) and transaction functions (external input, external output, and external inquires) from a functional perspective.

2) Object-Oriented metrics

Object-oriented software metrics have been developed, along with their software counterpart, based on object-oriented languages. Selected object-oriented metrics utilized in this work are explained below.
Cyclomatic complexity (CC) [11] is a count of the number of linearly independent paths through the source code which can be determined by equation (1). Fig. 1 depicts the graph of a sample program.

$$CC = E - N + 2P \tag{1}$$

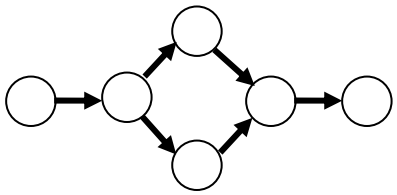

Fig. 1. A flow graph of example program.

where E is the number of edges of the graph, N is the number of nodes of the graph, and P is the number of connected components.
Weighted Methods per Class (WMC) [12] is defined as follows:

$$WMC = \sum_{i=1}^{n} c_i \tag{2}$$

where n is the number of methods and c_i is the complexity of each method.

Depth of Inheritance of the class (DIT) [12] is the maximum length which is counted from leaf class to root class of the inheritance tree. As a result, high value of DIT can reflect high design complexity.

Number of Children (NOC) [12] is the number of immediate subclasses in the class hierarchy tree. A large number of children can confuse the use of subclasses and call for additional access to a given method of the class.

Coupling between object classes (CBO) [12] is a count of the number of classes which is coupled with the class under investigation. If the class has two instant variables from two others classes, we count as two. Coupling between object classes is critical to modular design and prevents reuse of the class. A large number of the couples are sensitive to change in other parts of the design which, in turn, makes it difficult to maintain the software projects.

Response for a Class (RFC) [12] is a set of methods which can response to a message derived by an object of the class. If RFC is high, it will be difficult to test and debug.

Lack of Cohesion in Methods (LCOM) [12] is a count of the number of method pairs, whose similarity is zero minus the count of method pairs, having their similarity to be non-zero. If no method shares common variables, they have no similarity and the LCOM value will be zero. The value of the cohesion is low as method complexity is high.

III. ANALYSIS OF EFFORT ESTIMATION

The procedural analysis of effort estimation will be carried out in two stages, i.e., methodology and data analysis. Details on each stage are described below.

A. Methodology

We employed five the software projects for our proposed analysis methodology, namely, OPSOC, MOSTR, CW, PPTS, and KMSS. The Office of Permanent Secretary Operation Center **(OPSOC)** is a Decision Support System for manager consisting of 124 files. The Ministry of Science and Technology Research System **(MOSTR)** is an expert and research information archive system. The Collaborative Workspace **(CW)** is a project management system. The Public Project Training System **(PPTS)** is a government project biding system, and The Knowledge Management Support System **(KMSS)** serves as a shared knowledge repository. The first three projects were developed by the Government Information Technology Services (GITS), National Science and Technology Development Agency, while the remaining two are from Faculty of Political Science, Chulalongkorn University and Department of Health, Ministry of Public Health, respectively.

Project environments can be grouped by language and supporting platform, i.e., the OPSOC, MOSTR, and CW were written in C#.NET Framework v2.0, while the PPTS and KMMS were written in VB.NET Framework v2.0.

Having selected the projects for use in the experiment, we investigated a number of metric and cost estimation tools to assist in the analysis. Table III and IV outline sample tools in our consideration, but eventually we were in favor of free software as a baseline of measurement to avoid any subsequent legal complications that might incur.

To measure the quality of the above software projects, we have developed a software metric interface tool which incorporates a number of pertinent metrics, namely, SLOC, FP, DIT, NOC, WMC, RFC, CBO, LCOM, and CC. The tool makes use of several software cost estimation models to determine project effort, namely, COCOMO 81 Intermediate, COCOMO II Post-Architecture, COSYSMO, Walston-Felix, Bailey-Basili, Doty, SLIM, Albrecht-Gaffney, Kemerer, and Matson.

B. Data Analysis

All project data are preprocessed through the selected metrics and classified according to the operating characteristics of the project models. Analysis results are grouped into two main categories, namely, SLOC-based and FP-based to suit the inherent nature of the project data. Effort estimation, measured in man-month, is carried out in accordance with the underlying metrics. The formulas are listed below.

DIT based $$E = 4.11 + 0.1(DIT) \quad (3)$$

NOC based $$E = 4.11 + 0.1(NOC) \quad (4)$$

WMC based $$E = -2.939 + 0.028(WMC) \quad (5)$$

RFC based $$E = -9.223 + 0.013(RFC) \quad (6)$$

CBO based $$E = -7.274 + 0.025(CBO) \quad (7)$$

LCOM based $$E = 4.58 + 0.006(LCOM) \quad (8)$$

CC based $$E = -4.612 + 0.01(CC) \quad (9)$$

Mean Magnitude of Relative Error (MMRE) [13] is used to validate errors of estimation based on Magnitude of Relative Error (MRE), both of which are defined below.

$$MRE = \left| \frac{effort_{actual} - effort_{estimated}}{effort_{actual}} \right| \quad (10)$$

$$MMRE = \frac{1}{N} \sum_{i=1}^{N} MRE_i \quad (11)$$

TABLE III
EXAMPLES OF METRIC TOOLS

Tools	Explanation
CCCC	CCCC is a tool which analyzes C++ and Java files and generates a report on various metrics of the code.
Understand	Understand is used for source code analysis, reverse engineering, code visualization, and calculating code metrics.
OOMeter	The OOMeter software metric tool is an on-going initiative to measure the quality attributed of Java and C# source code and UML model, stored in XML format.
RSM	RSM is a source code metrics and quality analysis tool unlike any other on the market. RSM provides a standard method for analyzing C, ANSI C++, C# and Java source code across operating systems.
DynaMetrics	DynaMetrics introduces a new dynamic metric-based evaluation and analysis tool for Java and C++ software.

ckjm	ckjm calculates Chidamber and Kemerer object-oriented metrics by processing the bytecode of compiled Java files.
Analyst4j	Analyst4j encompasses two platforms, i.e., stand-alone and Eclipse plug-in, which support java language. Some of the features are composed of object-oriented Metrics, Maintainability Metrics, and Code Metrics.
SLOCCount	developed by David A. Wheeler, it is a set of tools for counting physical Source Lines of Code (SLOC) in a large number of languages of a potentially large set of programs.
SourceMonitor	provides metrics for source code written in C++, C, C#, VB.NET, Java, Delphi, VB6, and HTML by exporting metrics to XML or CSV.
NDepend	measures source code written in .NET code base and C#.
CCounter	Code Counter Pro (CCOunter) counts lines of code of programs for Windows platform.

TABLE IV
EXAMPLES OF SOFTWARE COST ESTIMATION TOOLS

Tools	Explanation
Costar	is a software estimation tools that supports COCOMO II.
SystemStar	estimates effort based on Dr. Ricardo Valerdi's COSYSMO model.
USC COCOMO 81	is an implementation of the 1981 COCOMO Intermediate model.
USC COCOMO II	predicts software development efforts, schedule, and costs based on COCOMO II Post-architecture model.
COST EXPERT	is a software cost estimation tool which can estimate COTS packaged implementations.

IV. EXPERIMENTAL RESULTS

The experimental results have many practical implications. Table V and VI depict some measured statistics obtained from conventional and object-oriented metrics, respectively. From Table V, measurement of project size via SLOC is 34.15% less than that of DSI based on C# (The PPTS and KMSS projects that were originally written in VB were subsequently converted to C#). The MRE of individual projects as measured by the metrics used are shown in Fig. 2-6. The plots reveal metrics and model applicability, e.g., project CW is best measured by SLOC using COCOMO 81. Effort measurements from all projects computed by different models in comparison with the actual values are depicted in Table VII. Note that Doty, Albrecht, and Kemerer models exhibit odd results due to their idiosyncrasies that are found to be unsuitable for small projects. Such a caveat is noticeable from the plots of Fig. 7 and 8 where SLOC exhibits higher concentration of estimates around the median than those of FP, which disperse across the range of measurement. Fig. 9 illustrates the resulting MMRE of all object-oriented models.

Table VIII summarizes all results based on the same metric measurement, i.e., SLOC, to arrive at a uniform comparison basis. It is apparent from Fig. 10 that the overall MMREs from individual model standpoint are almost the same except a slight deviation in the last three models. That signifies relatively indifferent measurability of the metrics. From the project standpoint of Fig. 11, the story is slightly different. The projects CW and OPSOC are statistically similar (from Table V and VI), whereas PPTS and KMSS exhibit the effects from language dependence. That means care must be taken in selecting the governing project model and applying proper metrics toward the measurement.

TABLE V
PROJECT MEASUREMENTS BASED ON SELECTED CONVENTIONAL METRICS

Metrics / Projects	CW	OPSOC	MOSTR	PPTS	KMSS
	C#	C#	C#	VB.NET	VB.NET
No. of files	173	124	62	138	151
KDSI	28.130	25.983	10.170	12.815	10.396
KSLOC	17.806	17.196	6.166	8.864	7.275
FP	295	291	127	175	141

TABLE VI
PROJECT MEASUREMENTS BASED ON SELECTED OBJECT-ORIENTED METRICS

Metrics/ Projects	Measure-ment stats	CW	OPSOC	MOSTR	PPTS	KMSS
No. of classes	Total	173	124	62	138	151
DIT	Average	0.913	1.355	0.984	1.225	0.934
	Total	158	168	61	169	141
NOC	Average	0.913	1.355	0.984	1.225	0.934
	Total	158	168	61	169	141
WMC	Average	7.832	5.758	9.742	3.652	3.748
	Total	1355	714	604	504	566
RFC	Average	15.983	18.242	18.339	14.717	13.689
	Total	2765	2262	1137	2031	2067
CBO	Average	8.893	8.234	9.774	6.304	6.185
	Total	1574	1021	606	870	934
LCOM	Average	27.908	30.105	23.161	3.949	1.351
	Total	4828	3733	1436	545	204
CC	Average	2.84	4.842	2.306	2.856	2.127
	Total	3848	3457	1386	1425	1202

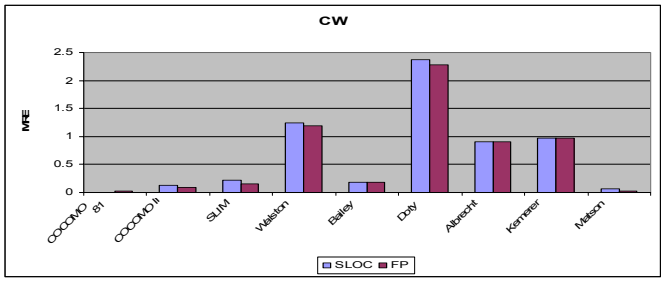

Fig. 2. MRE by model on CW project.

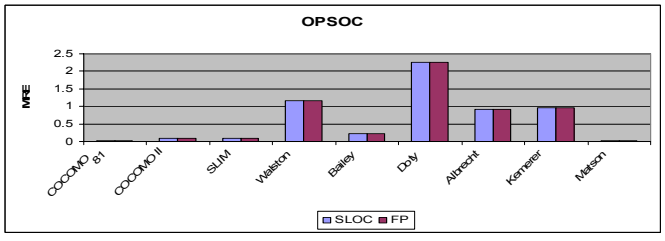

Fig. 3. MRE by model on OPSOC project.

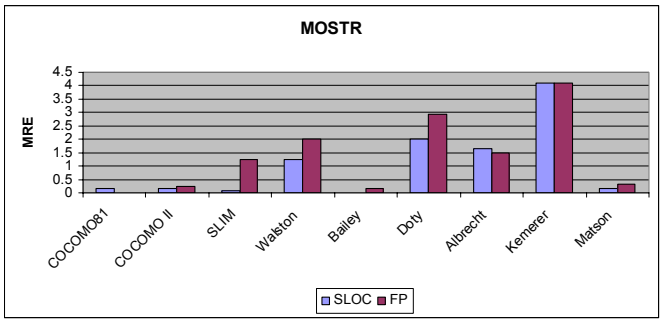

Fig. 4. MRE by model on MOSTR project.

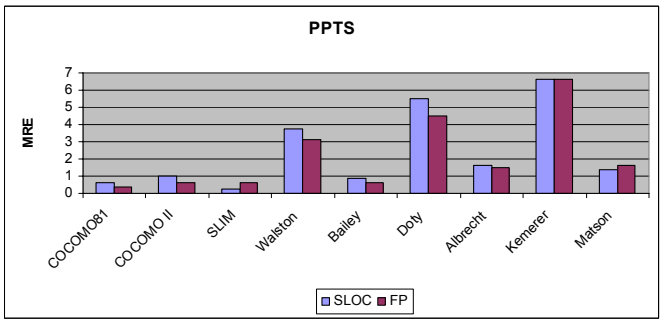

Fig. 5. MRE by model on PPTS project.

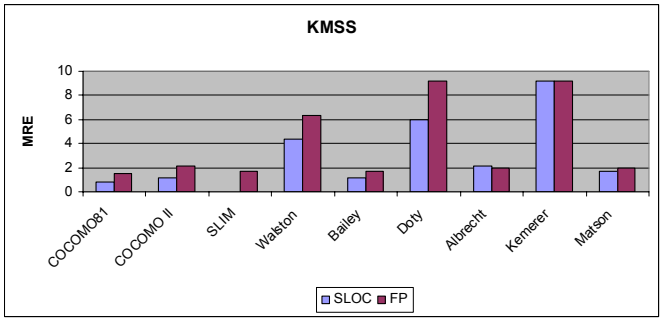

Fig. 6. MRE by model on KMSS project.

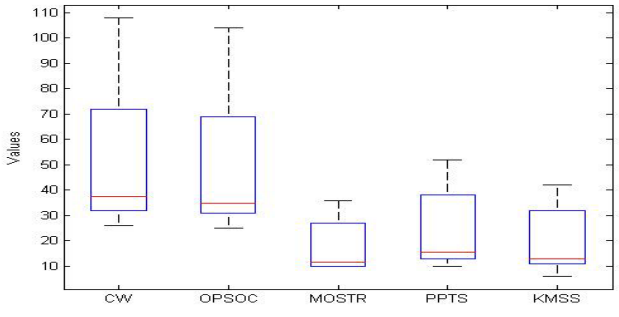

Fig. 7. project effort measurements based on SLOC.

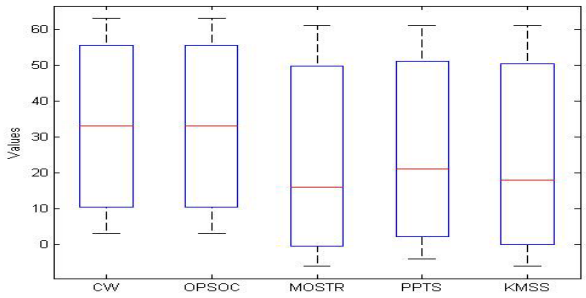

Fig. 8. project effort measurements based on FP.

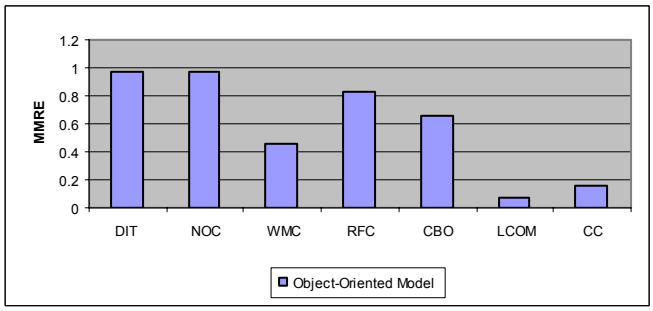

Fig. 9. MMRE of object-oriented models.

TABLE VII
PROJECT EFFORT MEASUREMENTS BASED ON SELECTED MODELS

Models / Projects	Metrics	CW	OPSOC	MOSTR	PPTS	KMSS
		C#	C#	C#	VB.NET	VB.NET
Actual value		32	32	12	8	6
COCOMO81	SLOC	32	31	10	13	11
COCOMO II		36	35	10	16	13
SLIM		39	35	11	10	6
Walston		72	69	27	38	32
Bailey		26	25	12	15	13
Doty		108	104	36	52	42
Albrecht	FP	3	3	-6	-4	-6
Kemerer		63	63	61	61	61
Matson		33	33	16	21	18

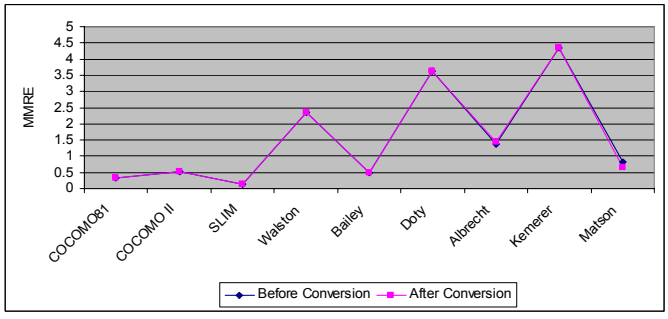

Fig. 10. MMRE by model.

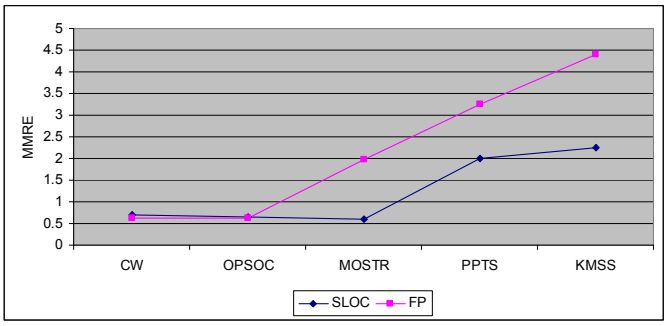

Fig. 11. MMRE by project.

TABLE VIII
COMBINED PROJECT EFFORT MEASUREMENTS AFTER CONVERSION

Models / Projects	Metrics	CW	OPSOC	MOSTR	PPTS	KMSS
Actual value		32	32	12	8	6
COCOMO81	SLOC	32	31	10	13	11
COCOMO II		36	35	10	16	13
SLIM		39	35	11	10	6
Walston		72	69	27	38	32
Bailey		26	25	12	15	13
Doty		108	104	36	52	42
Albrecht	SLOC* (FP)	3	3	-8	-5	-7
Kemerer		63	63	61	61	61
Matson		34	33	14	19	16

V. CONCLUSIONS

This research explores the interrelationship among different dimensions of software projects, namely, model, metrics, project size, and effort. Some inherent characteristics and idiosyncrasies of each dimension are unveiled in the analysis results. They are apparent from the most straightforward findings, such as SLOC and FP which are unsuitable for small size projects, to some complicated outcomes like LCOM and CC which surprisingly yield lowest MMRE measures on object-oriented models. From modeling standpoint, Albrecht model imposes certain limitations on its application that could result in negative measure; COSYSMO utilizes requirements, interfaces, critical algorithms, and operational scenarios as its computation bases; COCOTS is appropriate for plug-in scenarios. Nonetheless, smaller projects are worthy of conducting in-depth studies such as size-defect issue [14] rather than large projects. All in all, the benefits of these practical implications are two folds. First, project managers must understand the nature and relationship among various project elements such as estimation model, language, and metrics for effective application of effort estimation. Second, metrics and models are not the panacea of project management. Only the right tool, technique, and personnel of the right mix will get the project through. Our mission is to incorporate a machine learning technique to aid in finding the right mix for better project management.

ACKNOWLEDGMENTS

The authors wish to thank Government Information Technology Services (GITS), National Science and Technology Development Agency (NSTDA), Ministry of Science and Technology, for providing software project data and administrative support.

REFERENCES

[1] Sunita Chulani, Brad Clark, and Barry Boehm, "Calibration Results of COCOMO II.1997," the 22nd Annual Software Engineering Workshop, NASA Goddard Space Flight Center, December 1997.

[2] Jongmoon Baik, Barry Boehm, and Bert Steece, "Disaggregating and Calibrating the CASE Tool Variable in COCOMO II," IEEE Transactions on Software Engineering, vol. 28, no. 11, pp. 1009–1022, November 2002.

[3] Putnam, L.H., "A General Empirical Solution to the Macro Software Sizing and Estimating Problem," IEEE Transactions on Software Engineering, vol. se-4, no. 4, pp. 345–361, July 1978.

[4] C. E. Walston and C. P. Felix, "A method of Programming Measurement and Estimation," IBM Systems Journal, vol. 16, no. 1, pp. 54–73, 1977.

[5] John W. Bailey and Victor R. Basili, "A meta-model for software development resource expenditures," Proceedings of the 5th international conference on Software engineering, pp. 107–116, 1981.

[6] Albrecht, A.J. and Gaffney, J.E., Jr., "Software Function, Source Lines of Code, and Development Effort Prediction," IEEE Transactions on Software Engineering, vol. 9, no. 6, pp. 639–648, November 1983.

[7] Chris F Kemerer, "An empirical validation of software cost estimation models," IEEE Transactions on Software Engineering, vol. 30, no. 5, pp. 416–429, May 1987.

[8] Matson, J.E., Barrett, B.E., and Mellichamp, J.M., "Software development cost estimation using function points," IEEE Transactions on Software Engineering, vol. 20, no. 4, pp. 275–287, April 1994.

[9] A Carleton, et al., "Software Measurement for DoD Systems: Recommendations for Initial Core Measures," Technical Report CMU/SEI-92-019, ESC-TR-92-019, Software Eng. Inst., Carnegie Mellon University, Pittsburgh, September 1992.

[10] A.J. Albrecht, "Measuring application development productivity," Proceedings of the Joint SHARE/GUIDE/IBM Application Development Symposium, pp. 83–92, October 1979.

[11] Thomas J. McCabs, "A Complexity Measure," IEEE Transactions on Software Engineering, vol. se-2, no. 4, pp. 308–320, December 1976.

[12] Chidamber, S.R. and Kemerer, C.F., "A metrics suite for object oriented design," IEEE Transactions on Software Engineering, vol. 20, no. 6, June 1994, pp. 476–493.

[13] Keung, J.W., "Theoretical Maximum Prediction Accuracy for Analogy-Based Software Cost Estimation," the 15th Asia-Pacific on Software Engineering Conference 2008 (APSEC 08), pp. 495–502, December 2008.

[14] Koru, A.G., Zhang, D., El Emam, K., and Liu, H., "An Investigation into the Functional Form of the Size-Defect Relationship for Software Modules," IEEE Transactions on Software Engineering, vol. 35, no. 2, March/April 2009, pp. 293-304.