Studies in Software Cost Model Behavior: Do We Really Understand Cost Model Performance?

Karen Lum
Jet Propulsion Laboratory/
California Institute of
Technology
ktlum@jpl.nasa.gov

Jairus Hihn
Jet Propulsion Laboratory/
California Institute of
Technology
jhihn@jpl.nasa.gov

Tim Menzies
Lane Department of
Computer Science
West Virginia University
tim@menzies.us

Abstract[1]

While there exists extensive literature on software cost estimation techniques, industry practice continues to rely upon standard regression-based algorithms. These software effort models are typically calibrated or tuned to local conditions using local data. This paper cautions that current approaches to model calibration often produce sub-optimal models because of the large variance problem inherent in cost data and by including far more effort multipliers than the data supports. Building optimal models requires that a wider range of models be considered while correctly calibrating these models requires rejection rules that prune variables and records and use multiple criteria for evaluating model performance. The main contribution of this paper is to document a standard method that integrates formal model identification, estimation, and validation. It also documents what we call the large variance problem that is a leading cause of cost model brittleness or instability.

I. INTRODUCTION

In the 21st century, software managers have access to many different estimation tools. So how does one determine what is the right model to use over varying domains and organizations? This is a more serious problem than generally recognized because of the underlying large variance problem that is typically found in cost and effort data sets [1]. The variance problem causes model brittleness and makes it difficult to distinguish performance across various models. This is a pressing problem since effort estimates are often inaccurate. Early lifecycle effort estimates can be inaccurate by up to 400% [2, p310]. In 2001, the Standish group reported that, 53% of U.S. software projects ran over 189% of the original estimate [3]. In a study of NASA software development projects, the most frequently identified cause (71%) of cost overrun with the largest impact (35% contribution to observed cost growth) was *basic failures in planning, estimation & control* [4]. So the question becomes, how can we stop project managers and sometimes cost estimators from using the wrong (i.e. sometimes the worst) method just because it is convenient?

Clearly, better software estimation techniques are needed, as has been demonstrated by the various estimation techniques that have been proposed including clustering [5], neural networks [6], and case-based reasoning [7]. In industry, the most commonly-used formal techniques are parametric model and regression-based techniques such as those used in COCOMO 81 [2], COCOMO II [8], SEER-SEM [9], PRICE-S [10], and SLIM [11].

Unfortunately, there are very few published studies that empirically compare this diverse set of techniques. Those that have, such as Briand [12], Smith [13], Finnie [14], Gray [15], and Mair [6], are limited in scope. What is more usual are narrowly focused studies, such as those conducted by Kemerer [16], Lum [1], and Ferens [17] that test linear regression models in different environments. Given the need, why aren't there more studies comparing software effort models?

One possible explanation for these narrowly-focused studies is that comparing different effort models is fundamentally difficult. As shown in Table 1, standard software effort estimation data suffers from very

[1] The research described in this abstract was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government.

large performance deviations. The table shows thirty experiments where ten records (at random) were selected as a test set. Effort models were built on the remaining records, and then applied to the test set. Table 1 is sorted by the average deviation (in the final column) of performance scores. Note that the average deviation on the error can grow to over 300 times larger than the mean. Such large deviations make it difficult to distinguish the performance of different effort estimation models. Worse still, these deviations mean that managers cannot check if they are using the wrong (i.e. the worst) technique.

Table 1. Effort Modeling Results

	records	$T =$	average test error = mmre= $\sum_i^T \frac{\lvert predicted_i - actual_i \rvert}{actual_i * T}$		
source:part	$\lvert train \rvert$	$\lvert test \rvert$	mean	sd	$\frac{sd}{mean}$%
coc81:lang.mol	10	10	34	29	86
coc81:mode.org	13	10	32	27	87
coc81:lang.ftn	14	10	50	48	95
coc81:all	53	10	42	45	107
coc81:kind.max	21	10	47	51	107
coc81:mode.e	18	10	42	47	113
coc81:kind/min	11	10	47	66	139
nasa93:cat.missionplan	10	10	46	45	99
nasa93:cat.avionicsmonitor	20	10	43	47	107
nasa93:project.sts	28	10	68	142	206
nasa93:center.5	29	10	80	169	209
nasa93:year.1975	27	10	82	192	233
nasa93:fg.g	70	10	53	126	235
nasa93:mode.sd	59	10	58	149	254
nasa93:all	83	10	60	157	260
nasa93:year.980	28	10	81	211	260
nasa93:project.gro	13	10	56	168	296
nasa93:center.2	27	10	43	148	338
nasa93:mode.e	11	10	188	649	344

This article reports an extensive study using data mining techniques to identify the best performing models for varying specializations of records and variables. The tool developed to perform the analysis is called COSEEKMO, as the data we had available were all COCOMO data sets. The techniques described can easily be generalized to other standard models.

II. COSEEKMO

COSEEKMO is a tool for assessing different estimation models based upon COCOMO data sets using data mining techniques despite large deviations. COSEEKMO does this by generating numerous effort models from a specified data set using a range of techniques (local calibration, linear regression, model trees and the WRAPPER attribute selection algorithm [18]). Each technique is selected for its potential value in improving model prediction and reducing performance deviance. For example, the WRAPPER can cull the superfluous noisy attributes that confuse an effort model. The effort models found by COSEEKMO are assessed via rejection rules that cull the weaker models. Using the rules, the deviation in model performance seen in Table 1 can be significantly reduced. Without them, we show that certain classic effort estimation experiments cannot be reproduced.

The key elements of COSEEKMO are

1. The capability to repeatedly generate random data subsets containing records used for calibration/tuning and hold out or test sets to evaluate model performance (validation)
2. Programmable **Rejection Rules** for quantitatively assessing model performance
3. **Wrapper** for culling non Value-added cost drivers

Random data sets and test sets

The ability to randomly select a specified number of records as a test set and then use the remaining records for calibration/tuning, and to do this as many times as you want, is an important feature that cannot easily be performed manually by an estimator. Each randomly selected set is a different set than the next random set. This capability, in essence, gives the estimator a large dataset, from which to develop better, more accurate models.

Rejection Rules

The rejection rules are the core of COSEEKMO. In our own work, we revised the rules until they satisfied certain sanity checks.

In Experimental Methods for Artificial Intelligence, Cohen advises comparing the performance of a supposedly more sophisticated approach against a simpler "straw man" method [19, p81]. In COSEEKMO the straw man is a model based only on lines of code because clearly all other models are superfluous if estimates learned from just lines of code perform as well. Hence, we always ensure that one of the attribute subsets explored is just lines of code and effort.

Another important sanity check is that any new model should outperform COCOMO 81, or one should just use the off-the-shelf version of the model. To check this, the COCOMO 81 model is always explored, based on whether that software is an embedded system; a semi-detached system; or an organic system.

An important final sanity check was that the rules should able to reproduce at least one historical expert effort estimation study. Initially, COSEEKMO failed this sanity check since an early version of its rules could not reproduce Boehm's 1981 COCOMO 81 model from the original COCOMO 81 data set.

In addition to these sanity checks, the following model evaluation rules were used in this study:

- *Rule1* is a statistical test condoned by standard statistical textbooks. If a two-tailed t-test reports that the mean of two treatments is statistically different, then we can check if one mean is less than the other.
- *Rule2* (which checks for correlation) is a common check used at JPL: the best effort model tracks well between predicted and actual values. Without rule2, many parts of our data produce multiple survivors.
- *Rule 3* is added in case there is more than one model with the same average performance and R^2. The model with the smallest MMRE is selected

$$MMRE = \frac{100}{T}\sum_{i}^{T}\frac{\left|predicted_i - actual_i\right|}{actual_i}$$

- *Rule4* was added because PRED is a widely used performance measure for effort models. [20].

$$PRED(N) = \frac{100}{T}\sum_{i}^{T}\begin{cases} 1\, if\ MRE_i \le \frac{N}{100} \\ 0\, otherwise \end{cases}$$

- *Rule5* can be justified from Miller's work [21]. This rule rejects treatments that have similar performance, but use more attributes.

Different rules are applied because they measure different aspects of model performance. For example, Correlation, MMRE and PRED are subtly different performance measures. Overall, PRED measures how well an effort model performs while MMRE measures poor performance. A single large mistake can skew the MMREs and not affect the PREDs. Shepperd and Schofield comment that:

> MMRE is fairly conservative with a bias against overestimates while Pred(30) will identify those prediction systems that are generally accurate but occasionally wildly inaccurate [7, p736].

Wrapper

COSEEKMO's attribute pruning method is called the "WRAPPER" [22]. Starting with the empty set, the WRAPPER adds some combinations of columns (columns contain data on effort, size, and fifteen COCOMO 81 cost drivers) and asks some learner (in our case, the LC method discussed below) to build an effort model using just those columns. The WRAPPER then grows the set of selected attributes and checks if a better model comes from learning over the larger set of attributes. The WRAPPER stops when there are no more attributes to select, or there has been no significant improvement in the learned model for the last five additions (in which case, those last five additions are deleted). Technically speaking, this is a forward select search with a "stale" parameter set to 5.

COSEEKMO uses the WRAPPER since experiments by other researchers strongly suggest that it is superior to many other attribute pruning methods. For example, Hall and Holmes [23] compare the WRAPPER to several other attribute pruning methods including principal component analysis (PCA- a widely used technique). Column pruning methods can be grouped according to:

- Whether or not they make special use of the target attribute in the data set such as “development cost”;
- Whether or not they use the target learner as part of their pruning.

PCA is unique since it does not make special use of the target attribute. WRAPPER is also unique, but for different reasons: unlike other pruning methods, it does use the target learner as part of its analysis. Hall and Holmes found that PCA was one of the worst performing methods (perhaps because it ignored the target attribute) while WRAPPER was the best (since it can exploit its special knowledge of the target learner). WRAPPER is thorough but, theoretically, it is quite slow since (in the worst case), it has to explore all subsets of the available columns. However, all the data sets in this study are small enough to permit the use of the wrapper.

III. DATA

The data used in the analysis is from the original 63 records in the COCOMO 81 data set and a historical NASA data set we refer to as NASA93 as it has 93 flight and ground records form multiple NASA Centers that completed from the late 1970’s through the late 1980’s. The NASA93 data has been in the public domain for many years but few have been aware of it. It can now be found at the PROMISE (Predictor Models in Software Engineering) web site. [2] PROMISE is an organization of software engineers working to organize data sets that can be used to verify existing studies and to make data available for future research. To participate in a PROMISE workshop one must promise to make their data available to the PROMISE repository.

In this study, COSEEKMO built effort estimators using all or some part of two COCOMO 81 data sets (nasa93 and coc81). Each part selected some subset of the total records. The parts and data sets are described in more detail in Figure 1.

Data	*Coc81:*	has 63 records in the COCOMO 81 format
	Nasa93:	has 93 NASA records in the COCOMO 81 format
Subsets/Stratification Categories	*All:*	selects all records from a particular source; e.g.. "coc81_all" and "nasa93_all"
	Category:	is a NASA-specific designation selecting the type of project; e.g. avionics, data capture, etc.
	Fg:	selects either "f" (flight) of "g" (ground) software
	Kind:	selects records relating to the development platform; max = mainframe and mic = microprocessor
	Lang:	selects records about different development languages
	Center:	*nasa93* designation selecting records relating to where the software was built
	Project:	*nasa93* designation selecting records relating to the name of the project
	Mode:	selects records relating to different COCOMO 81 development modes; *org*, *sd*, and *e* are short for organic, semi-detached, and embedded (respectively)
	Type:	selects different COCOMO 81 designations and include "bus" (for business application) or "sys" (for system software)
	Year:	is a *nasa93* term that selects the development years, grouped into units of five; e.g. 1970, 1971, 1972, 1973, 1974 are labeled "1970"

Figure 1. Data and subsets used in this study.

IV. EXPERIMENTS AND RESULTS

When an analysis contains tens of thousands of runs over hundreds of experiments, there are many different ways of looking at the results. In this paper the focus has been narrowed by only addressing

[2] http://promise.site.uottawa.ca/SERepository/

three main issues that we deemed would be of most interest to cost model developers and cost estimators:

- Local Calibration and Stratification
- Finding the Best Model
- Cost Driver Instability

Local Calibration and Stratification

The two methods most commonly used to tune models to local environments are

- **Local calibration (or LC);** i.e. using local data to estimate the 'a' and 'b' parameters of the standard COCOMO equation : $effort(personmonths) = a * \left(KLOC^{b}\right) * \left(\prod_{j} EM_{j}\right)$
- **Stratification**; i.e. given a database of past projects, and a current project to be estimated, restrict local calibration to just those records from similar projects.

These two approaches can easily be shown to fall short of what should be expected from a best model. If stratification improved performance, then subsets of the data should usually generate better models with lower error rates than models learned from all the data. This is not necessarily the case as can be seen in Figure 2:

- The dashed horizontal lines of Figure 2 shows the error rate of models learned from *all* data from the two sources.
- The crosses of Figure 2 show error rates seen in models learned from *subsets* of the data.
- Seven subsets fall below the lines; i.e. those subsets generate models with *lower* and *better* error rates;
- But an equal number fall above the lines; i.e. they generate *higher* and *worse* error rates.

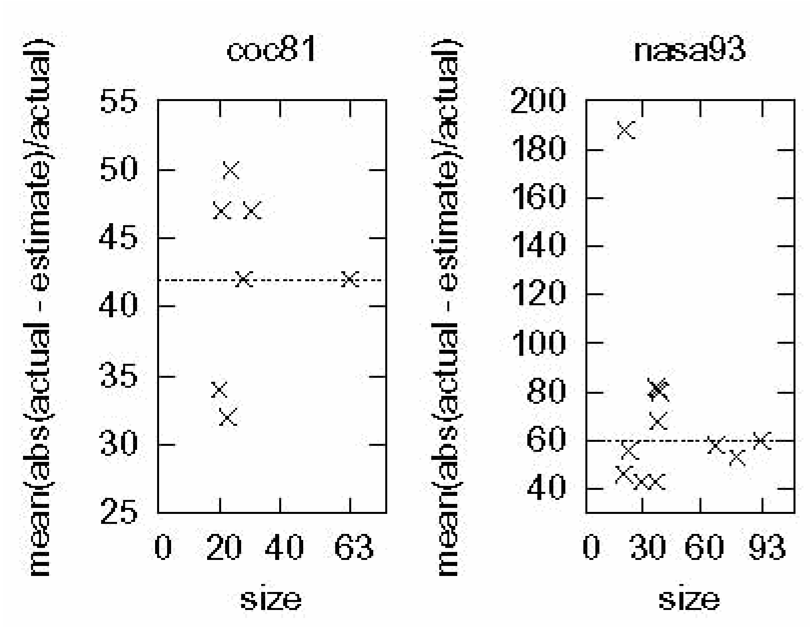


Figure 2. The plots show mean performance error (i.e. |(predicted − actual)|/actual) found after 30 experiments with each subset. In each experiment, ten records were selected at

random to be a test set. Effort models were built on the remaining records using COCOMO's local calibration method [2, p526-529], then applied to the test set. The horizontal lines show the mean error performance using all records in the sources. The crosses show the mean error performance seen in models learned from subsets of that data. Crosses below/above the lines indicate models performing better/worse (respectively) than models built from all the data.

In this study we looked at 207 possible stratifications, only four subsets generate better effort models. It was found that in 98% of the cases stratification was not beneficial for our data. This is not to say stratification is always useless. Rather, stratifications must be selected and assessed with greater care than has been practiced to date.

A similar conclusion was found for local calibration. While LC sometimes produces the best models, often it does not. Figure 3 compares standard LC with the multiple algorithms contained in our COSEEKMO tool. Note while standard LC sometimes does as well as anything else, COSEEKMO's algorithms yielded models with overall smaller mean error and much smaller standard deviation on the model error. Reducing the error deviation is most important: without it, the results generated from effort modeling are highly unreliable.

The results indicate that effort modeling needs to be much more than just LC and stratification. In our view, LC and stratification are just two members of a large set of modeling methods. Our proposed effort modeling methodology is to try all the methods, then select the one that works best for the local domain.

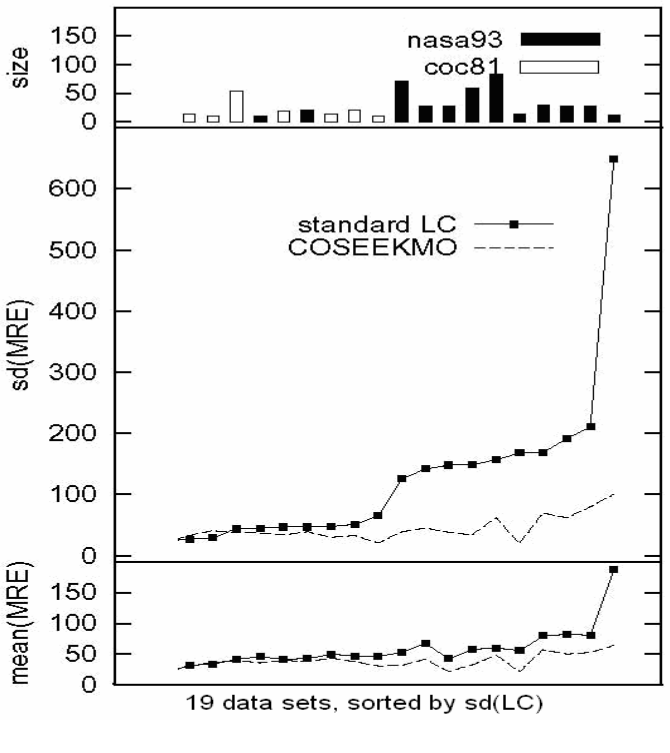


Figure 3. The top plot shows the number of projects in 27 subsets of our two data sources. The middle and bottom plots show the standard deviation and mean in performance error. Data subsets are sorted by the error's standard deviation. Effort models were learned via either standard LC or COSEEKMO.

Finding the Best Model

A summary of the best models as identified by COSEEKMO is displayed in Table 2. The stratifications shown are for all data subsets of coc81 and nasa93 with 20 or more records. There are five observations we would like to draw to the reader's attention.

Firstly, COSEEKMO's rules were adjusted until they concurred with Boehm's 1981 analysis. Hence, there are no surprises in the coc81 results. Coc81 did not require any of COSEEKMO's advanced modeling techniques (model trees, or the WRAPPER); i.e. this study found nothing better than the methods published in 1981 for processing the COCOMO 81 data. For example:

- The embedded and organic "a" and "b" values worked best for coc81 embedded and organic systems (row 3 and row 6 of Table 2).
- Local calibration was called only once for coc81.all (row 5 of Table 2) and this is as expected. Coc81.all is a large mix of different project types so it is inappropriate to use canned values for embedded, semi-detached, or organic systems

Table 2 Survivors from *Rejection Rule 1,2,3,4,5*.

		Records		Treatment			Results		
row	source:part	T=|*train*|	T=|*test*|	Numbers	|*Subset*|	Learn	Mean PRED(30)	MMRE mean	MMRE Sd
1.	coc81:kind.min	11	10	precise	17	e	60	31	21
2.	coc81:lang.ftn	14	10	precise	17	sd	42	44	30
3.	coc81:mode.e	18	10	precise	17	e	46	40	34
4.	coc81:kind.max	21	10	precise	17	e	52	38	33
5.	coc81:all	53	10	precise	17	LC	50	40	37
6.	coc81:mode.org	13	10	precise	17	org	62	32	33
7.	coc81:lang.mol	10	10	precise	17	sd	56	36	41
8.	nasa93:project.Y	13	10	precise	16	LC	78	22	20
9.	nasa93:category.missionplanning	10	10	rounded	17	e	50	36	37
10.	nasa93:category.avionicsmonitoring	20	10	precise	8	M5P	53	38	39
11.	nasa93:mode.sd	59	10	rounded	7	LC	62	33	34
12.	nasa93:project.X	28	10	precise	17	e	42	42	45
13.	nasa93:fg.g	70	10	rounded	10	LSR	65	32	39
14.	nasa93:center.5	29	10	precise	12	LC	43	57	70
15.	nasa93:year.1975	27	10	precise	11	LSR	52	50	62
16.	nasa93:all	83	10	rounded	14	LSR	43	48	62
17.	nasa93:year.1980	28	10	precise	16	LC	53	53	80
18.	nasa93:mode.e	11	10	precise	17	e	42	64	100
19.	nasa93:center.2	27	10	precise	17	LC	83	22	38

Secondly, COSEEKMO can dramatically reduce the deviation in model performance. In the results of Figure 3, nasa93's normalized deviations from standard LC had a median of 254% and, in 10/12 cases, was over 200%. In the results of Table 2, generated by COSEEKMO, nasa93's normalized deviations have a median value of 122% and, in no case was it over 200%.

Thirdly, in six cases (rows 9,11,13,16 of Table 2), COSEEKMO found the precise COCOMO numerics were superfluous and that the rounded values sufficed. That is, for those data sets, the precise COCOMO numerics were an over-specification of the effects of attributes on the total effort.

Fourthly, in 8 cases (rows 8,10,11,13,14,15,16,17 of Table 2) the WRAPPER discarded, on average, five to six attributes, and sometimes many more (e.g. in row 10,11 of Table 2 the surviving models discarded nearly half the attributes). This result, plus the last one, raises concerns for those that propose changes to business practices based on the precise COCOMO numerics published in the standard COCOMO texts [2], [8]. Before using part of an estimation model to make a business case (e.g. such as debating the merits of sending analysts to specialized training classes), it is advisable to check if that part of the standard COCOMO model is culled by better effort models.

Fifthly, many of the results in Table 2 use non-standard effort estimation methods. As mentioned above, in 8 experiments, the WRAPPER was useful. Also, in four cases, the best effort models were generated using linear regression (rows 13,15,16 of Table 2) or model trees (row 10 of Table 2). That is, standard effort estimation methods are not optimal in all cases and should be augmented with other techniques

To summarize this section our results indicate that no one approach to tuning a model is always the best approach. Sometimes one can use the model right out of the box, sometimes local calibration is sufficient, and sometimes a full regression analysis needs to be performed to obtain optimal results. The good news is that the standard functional form shown below is virtually always selected as indicated by the non-standard model M5P being selected only once.

$$effort(personmonths) = a * \left(KLOC^{b}\right) * \left(\prod_{j} EM_{j}\right)$$

Cost Driver Instability

As we ran the experiments we began to notice that the number of significant cost drivers in the 'best' models, as reported in Table 2, was constantly changing and not in any systematic manner. Obviously, this started to raise a number of questions. Is there a consistent set of cost drivers that are significant across all stratifications of the data? For any given stratification, is a cost driver always significant or not, or is it sometimes in and sometimes out? Which cost drivers get dropped or kept the most?

In the machine learning field the reduction of variables is called feature subset selection. One of the standard models is WRAPPER, which is included in COSEEKMO. While an extensive analysis is planned for the future, reported here are the WRAPPER results run as part of finding the best performing model using local calibration (LC). This is relevant because local calibration seems to be the most frequently used method for tuning commercial and industry cost models like COCOMO.

COSEEKMO applies WRAPPER to perform feature subset selection after completing the local calibration experiments to see if a better model can be found by reducing the number of cost drivers. Again the procedure is to randomly select a test set and then randomly select from the remaining records to construct the calibration set. The construction of the calibration sets is repeated 10 times and tested against the test set. We call this an experiment. Then a new test set is selected and another feature subset selection analysis was repeated 10 times on the remaining dataset. This goes on for a total of 30 experiments. Within each experiment, the number of times a cost driver was selected out of the 10 times is documented. The 30-experiments were performed for most of the subsets of data in Table 1. *Lang* and *Kind* were skipped because the types of languages and platforms were outdated and deemed not of interest.

For each trial, the cost driver can be selected anywhere from 0 to 10 out of 10 times. Cost driver stability is indicated when a cost driver was either never selected or was selected 10 out of 10 times. Unfortunately, there were many experiments in which the results were mixed. To determine if the degree of inconsistency in the results was significant, a statistical test had to be identified. Given the data was binary, selected vs. non-selected, and that we are counting the number of times something occurred, the data fits the standard description for using a median test. A 95% confidence interval for the median was computed based on the rank order of the data and a test was performed for each cost driver in each subset to determine if the number of times a cost driver was selected was significantly different from 10. The results are summarized in Table 3.

In Table 3, a blue filled circle indicates the median of the 30 trials is not significantly different than 10. An orange unfilled circle indicates the median was not significantly different from 9. At the time, we were unable to test to see whether there was a difference between a cost driver being selected 10 out of 10 times versus 9 out of 10 times. Therefore, if the median of the 30 trials was not significantly different than 9, they may or may not be significant to the subset. The grayed-out boxes in Table 3 indicate the cost driver was not significant for that subset and had a median confidence interval that did not include 9 or 10.

The results are significant and indicated by the fact that COSEEKMO reproduces Dr. Boehm's original COCOMO model when using his data. It makes sense that for **coc81_all** that all fifteen cost drivers are significant, since the data is a mix of various modes, categories, and years. Overall the stratifications (embedded, semi-detached, and embedded) also have the most significant cost drivers compared to the other stratifications. These unsettling results, which we strongly believe are also correct results, are that for the NASA data set and its stratifications, there is no clear pattern. It can be easily seen in Table 3, that a cost driver in one subset might be significant, whereas, it is not significant in another subset. For example, the *pcap* cost driver is significant for the mission planning subset, but not for the avionics monitoring subset. Why should there be a difference in this case?

Until we have time to pursue a more detailed analysis, what we are recommending is that cost estimators use these results to guide them in their model development and cost analysis. The results indicate that the cost drivers for projects from 1970-1990 that are most likely to be significant are acap, time, cplx, aexp, virt, data, turn, rely, and stor. The cost drivers that are unlikely to improve model performance are pcap, vexp, lexp, modp, tool, sced. It is expected for more contemporary data that stor and time would drop out because there are fewer computer constraints these days and modp may become more significant, but only if organizations are forward-looking and trying to engage new design methods and programming languages.

V. CONCLUSION

Current definitions of best practice for industrial effort estimation using linear regression often include stratification and local calibration. We have shown here that those definitions need to be extended. Sometimes, LC and stratification are indeed the best effort modeling methods. However, often they aren't and other methods should be used instead.

Distinguishing between rival modeling methods is problematic. Often, effort models must be learned from a very small number of records and generalization from limited experience is an inherently under-constrained task. Consequently, the deviations in model performance can be so large (see Figure 3) that standard statistical methods (e.g. t-tests) can't distinguish between the performance of rival methods. In our view, this problem of large deviances has not been adequately acknowledged or addressed. Furthermore, we believe it is the large variance problem that underlies why cost modeling and estimation is more art than engineering.

Little can be done about the size of the training or test data; small sample sizes are common in effort estimation and must be somehow managed. COSEEKMO manages learning from small sample sizes via a set of model generation techniques: local calibration, linear regression, model trees, and the WRAPPER. Each technique was selected to address one or more of the possible causes of performance variability.

COSEEKMO constrains estimation model generation via rejection rules. These rules model the heuristic model selection criteria (art aspects of cost modeling) used by effort modelers. To the best of our knowledge, these rules have not been seriously studied prior to this article. By explicitly representing them, the rules can now be audited, reviewed, and revised.

Another major result is that our experiments indicate that most cost models have far too many cost drivers, which means our models tend to be over-specified and have too many switches or knobs. More parsimonious models would provide clearer results and be reproducible, which would help to move our profession from being more art to an engineering/science discipline. Of course, this might make it harder for us to spin the results and continue to keep our sponsor's off balance so they will keep approving those low-ball estimates.

Table 3. Number of Significant Cost Drivers in each Subset based on 95% Median Significance Test

	COCOMO 81 Cost Drivers															Number of Significant Cost Drivers
Data Subset	acap	time	cplx	aexp	virt	data	turn	rely	stor	lexp	pcap	modp	vexp	sced	tool	
coc81_all	○	●	●	●	●	●	●	●	●	●	●	●	●	●	●	15
coc81_mode_embedded	○	●	○	○	●	○	○	○	○	●		●	●	●	●	14
coc81_mode_organic	●	●	○	●	●	●	●		○		●	●	●	●	●	13
nasa93_all	●	●		●	●	●	●	●	●							8
nasa93_mode_embedded	○	●	●		●	●	●	●	●	○	○			●		11
nasa93_mode_semidetached	●			●									○			3
nasa93_fg_ground	●		○	●						●	○					5
nasa93_category_missionplanning	○	●	●			●	●	●			●	○		○		9
nasa93_category_avionicsmonitoring	●			○								●	○	○	○	6
nasa93_year_1975	●	●	●	●	●	●		●	●	○	○					10
nasa93_year_1980	●	●	●	○	●	●	●	●	●					●	○	11
nasa93_center2	●	●	●	●	●	○	●	○	●	●	●	●	●		●	14
nasa93_center5		●	●	○	●	●	○	●	●	○						9
nasa93_project_gro	○	○	●	○	●		●	○	○	●	○	●	●		○	13
nasa93_project_sts		●	●		●	●	●	●	●							7
Usually Significant	5	1	3	5	0	2	2	3	3	3	4	1	2	2	3	
Always Significant	8	11	9	7	11	9	9	8	8	5	4	6	5	5	4	
Total Number of Significant Occurrences	**13**	**12**	**12**	**12**	**11**	**11**	**11**	**11**	**11**	**8**	**8**	**7**	**7**	**7**	**7**	

Legend:

● = Not significantly different than 10 at a 95% Confidence Interval

○ = Not significantly different than 9 or greater at a 95% Confidence Interval

REFERENCES

[1] K. Lum, J. Powell, and J. Hihn, “Validation of spacecraft cost estimation models for flight and ground systems,” in ISPA Conference Proceedings, Software Modeling Track, May 2002.

[2] B. Boehm, *Software Engineering Economics*. Prentice Hall, 1981.

[3] “The Standish Group Report: Chaos 2001,” 2001, available from http: //standishgroup.com/sample research/PDFpages/extreme chaos.pdf.

[4] J. Hihn and H. Habib-agahi, “Identification and measurement of the sources of flight software cost growth,” in *Proceedings of the 22nd Annual Conference of the International Society of Parametric Analysts (ISPA), Noordwijk, Netherlands*, 8-10 May 2000.

[5] M. Garre, M. S. J.J. Cuadrado-Gallego, M. Charro, and D. Rodriguez, “Segmented parametric software estimation models: Using the em algorithm with the isbsg 8 database,” in *27th International Conference on Information Technology Interfaces. ITI 2005, Dubrovnik, Croatia*, 2005.

[6] C. Mair, G. Kadoda, M. Lefley, K. Phalp, C. Schofield, M. Shepperd, and S. Webster, “An investigation of machine learning based prediction systems,” *The Journal of Systems and Software*, vol. 53, no. 1, pp. 23–29, 2000. [Online]. Available: citeseer.ist.psu.edu/mair99investigation.html

[7] M. Shepperd and C. Schofield, “Estimating software project effort using analogies,” IEEE Transactions on Software Engineering, vol. 23, no. 12, November 1997, available from http://www.utdallas.edu/~rbanker/SE XII.pdf.

[8] B. Boehm, E. Horowitz, R. Madachy, D. Reifer, B. K. Clark, B. Steece, A. W. Brown, S. Chulani, and C. Abts, *Software Cost Estimation with Cocomo II*. Prentice Hall, 2000.

[9] R. Park, “The central equations of the price software cost model,” in *4th COCOMO Users Group Meeting*, November 1988.

[10] R.Jensen, “An improved macrolevel software development resource estimation model,” in *5th ISPA Conference*, April 1983, pp. 88–92.

[11] L. Putnam and W. Myers, Measures for Excellence. Yourdon Press Computing Series, 1992.

[12] L. Briand, T. Langley, and I. Wieczorek, “A replicated assessment and comparison of common software cost modeling techniques,” in Proceedings of the 22nd International Conference on Software Engineering, Limerick, Ireland, 2000, pp. 377–386.

[13] A. Smith and A. Mason, “Cost estimation predictive modeling: Regression versus neural network,” *The* Engineering Economist, vol. 42, no. 2, pp. 137–161, 1997.

[14] G. Finnie, G. Wittig, and J. Desharnais, “A comparison of software effort estimation techniques: Using function points with neural networks, case-based reasoning and regression models,” Journal of Systems and Software, vol. 39, no. 3, pp. 281–289, 1997.

[15] A. Gray and S. MacDonnell, “A comparison of techniques for developing predictive models of software metrics,” *Information and Software Technology*, vol. 39, 1997.

[16] C. Kemerer, “An empirical validation of software cost estimation models,” *Communications of the ACM*, vol. 30, no. 5, pp. 416–429, May 1987.

[17] D. Ferens and D. Christensen, “Calibrating software cost models to Department of Defense Database: A review of ten studies,” *Journal of Parametrics*, vol. 18, no. 1, pp. 55–74, November 1998.

[18] R. Kohavi and G. H. John, “Wrappers for feature subset selection,” *Artificial Intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997. [Online]. Available: citeseer.nj.nec.com/kohavi96wrappers.html

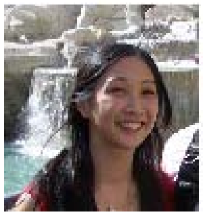
[19] P. Cohen, *Empirical Methods for Artificial Intelligence*. MIT Press, 1995.

[20] S. Chulani, B. Boehm, and B. Steece, “Bayesian analysis of empirical software engineering cost models,” *IEEE Transaction on Software Engineering*, vol. 25, no. 4, July/August 1999.

[21] A. Miller, *Subset Selection in Regression (second edition)*. Chapman & Hall, 2002.

[22] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.

[23] M. Hall and G. Holmes, “Benchmarking attribute selection techniques for discrete class data mining,” *IEEE Transactions On Knowledge And Data Engineering*, vol. 15, no. 6, pp. 1437– 1447, 2003.



Karen Lum is a senior cost analysis at the Jet Propulsion Laboratory, involved in the collection of software metrics, and the development of software cost estimating relationships. She has a MBA in Business Economics and a Certificate in Advanced Information Systems from the California State University, Los Angeles. She has a BA in Economics and Psychology from the University of California at Berkeley. She is one of the main authors of the JPL Software Cost Estimation Handbook. Publications include Best Conference Paper for ISPA 2002: Validation of Spacecraft Software Cost Estimation Models for Flight and Ground Systems. email: ktlum@jpl.nasa.gov

Jairus Hihn is a Principal Member of the Engineering staff at the Jet Propulsion Laboratory and is currently the manager for the Software Quality Improvement Projects Measurement Estimation and Analysis Element, which is establishing a laboratory wide software metrics and software estimation program at JPL. M&E's objective is to enable the emergence of a quantitative software management culture at JPL. He has a Ph.D. in Economics from the University of Maryland. He has been developing estimation models and providing software and mission level cost estimation support to JPL's Deep Space Network and flight projects since 1988. He has extensive experience in simulation and Monte Carlo methods with applications in the areas of decision analysis, institutional change, R&D project selection cost modeling, and process models. email jhihn@jpl.nasa.gov

Tim Menzies is an associate professor at the Lane Department of Computer Science at the University of West Virginia (USA), and has been working with NASA on software quality issues since 1998. He has a CS degree and a PhD from the University of New South Wales. His recent research concerns modeling and learning with a particular focus on light weight modeling methods. His doctoral research aimed at improving the validation of, possibly inconsistent, knowledge-based systems in the QMOD specification language. He also has worked as an object-oriented consultant in industry and has authored over 150 publications and served on numerous conference and workshop programs and well as guest editor of journal special issues. email tim@menzies.us