Selecting Best Practices for Effort Estimation

Tim Menzies, *Member*, *IEEE*, Zhihao Chen, Jairus Hihn, and Karen Lum

Abstract—Effort estimation often requires generalizing from a small number of historical projects. Generalization from such limited experience is an inherently underconstrained problem. Hence, the learned effort models can exhibit large deviations that prevent standard statistical methods (e.g., t-tests) from distinguishing the performance of alternative effort-estimation methods. The COSEEKMO effort-modeling workbench applies a set of heuristic *rejection rules* to comparatively assess results from alternative models. Using these rules, and despite the presence of large deviations, COSEEKMO can rank alternative methods for generating effort models. Based on our experiments with COSEEKMO, we advise a new view on supposed "best practices" in model-based effort estimation: 1) Each such practice should be viewed as a *candidate* technique which *may or may not* be useful in a particular domain, and 2) tools like COSEEKMO should be used to help analysts explore and select the best method for a particular domain.

Index Terms—Model-based effort estimation, COCOMO, deviation, data mining.

1 Introduction

Effort estimation methods can be divided into *model-based* and *expert-based* methods. Model-based methods use some algorithm to summarize old data and make predictions about new projects. Expert-based methods use human expertise (possibly augmented with process guidelines, checklists, and data) to generate predictions.

The list of supposed "best" practices for model and effort-based estimation is dauntingly large (see Fig. 1). A contemporary software engineer has little guidance on which of these list items works best, which are essential, and which can be safely combined or ignored. For example, numerous studies just compare minor algorithm changes in model-based estimation (e.g., [1]).

Why are there too few published studies that empirically check which methods are "best"? The thesis of this paper is that there is something fundamental about effort estimation that, in the past, has precluded comparative assessment. Specifically, effort estimation models suffer from very *large performance deviations*. For example, using the methods described later in this paper, we conducted 30 trials where 10 records (at random) were selected as a test set. Effort models were built on the remaining records and then applied to the test set. The deviations seen during testing were alarmingly large. In one extreme case, the standard deviation on the error was enormous (649 percent; see the last row of Fig. 2).

These large deviations explain much of the contradictory results in the effort estimation literature. Jorgensen reviews 15 studies that compare model-based to expert-based estimation. Five of those studies found in favor of expert-based methods, five found no difference, and five found in favor of model-based estimation [2]. Such diverse conclusions are to be expected if models exhibit large deviations, since large deviations make it difficult to distinguish the performance of different effort estimation models using standard statistical methods (e.g., t-tests). *If the large deviation problem cannot be tamed, then the expert and model-based effort estimation communities cannot comparatively assess the merits of different supposedly best practices.*

The rest of this paper offers a solution to the large deviation problem. After a discussion of the external validity and some of the background of this study, some possible causes of large deviations will be reviewed. Each cause will suggest operations that might reduce the deviation. All these operations have been implemented in our new COSEEKMO toolkit. The design of COSEEKMO is discussed and its performance is compared with the Fig. 1 results. It will be shown that COSEEKMO's operators reduce large deviations and also improve mean errors for model-based estimation.

2 External Validity

As with any empirical study, our conclusions are biased according to the analysis method. In our case, that bias expresses itself in four ways: biases in the method, biases in the model, biases in the data, and biases in the selection of data miners (e.g., linear regression, model trees, etc.).

Biases in the method. The rest of this paper explores model-based, not expert-based, methods. The comparative evaluation of model-based versus expert-based methods must be left for future work. Before we can compare any effort estimation methods (be they model-based or expert-based), we must first tame the large deviation problem. For more on expert-based methods, see [2], [6], [9], [16]

Biases in the model. This study uses COCOMO data sets since that was the only public domain data we could access. Nevertheless, the techniques described here can easily be generalized to other models. For example, here, we use

- *T. Menzies is with the Lane Department of Computer Science, West Virginia University, Morgantown, WV 26506-610. E-mail: tim@menzies.us.*
- *Z. Chen is at 941 W. 37th Place, SAL 337, Los Angeles, CA 90089. E-mail: chenscott@gmail.com.*
- *J. Hihn and K. Lum are with the Jet Propulsion Laboratory, 4800 Oak Grove Drive, Pasadena, CA 91109-8099. E-mail: {jhihn, ktlum}@mail.jpl.nasa.gov.*

Manuscript received 20 Feb. 2006; revised 26 June 2006; accepted 20 Aug. 2006; published online DD Mmmm, YYYY.
Recommended for acceptance by P. Clements.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-0041-0206.

According to Jorgensen [2], expert-based best practices include:

1. evaluate estimation accuracy, but avoid high evaluation pressure;
2. avoid conflicting estimation goals;
3. ask the estimators to justify and criticize their estimates;
4. avoid irrelevant and unreliable estimation information;
5. use documented data from previous development tasks;
6. find estimation experts with relevant domain background;
7. estimate top-down and bottom-up, independently of each other;
8. use estimation checklists;
9. combine estimates from different experts and estimation strategies;
10. assess the uncertainty of the estimate;
11. provide feedback on estimation accuracy; and,
12. provide estimation training opportunities.

According to Boehm [3], [4]; Chulani [5], [6]; Kemerer [7]; Stutzke [8]; Shepperd [9]; our own work [10]–[12]; and a recent tutorial at the 2006 International Conference of the International Society of Parametric Analysts [13], best practices for model-based estimation include at least the following:

13. *Reuse regression parameters* learned from prior projects on new projects;
14. *Log-transforms* on costing data before performing *linear regression* to learn log-linear effort models;
15. *Model-tree* learning to generate models for non-linear relationships;
16. *Stratification*, i.e. given a database of past projects, and a current project to be estimated, just learn models from those records from similar projects;
17. *Local calibration*, i.e. tune a general model to the local data via a small number of special tuning parameters;
18. *Hold-out* experiments for testing the learned effort model [10];
19. Assessing effort model uncertainty via the *performance deviations* seen during the hold-out experiments of item #17;
20. *Variable subset selection* methods for minimizing the size of the learned effort model [11], [12], [14], [15].

This separation of model-based and expert-based methods is not a strict division since some practices fall into both categories: e.g. #4 & #20 are similar as are #10 & #19. Also, one way to view model-based methods is that they seek algorithms to make maximal use of #5. Further, some research actively tries to combine the two approaches:

21. Shepperd's *case-based reasoning* tools [9] explore algorithmic methods for emulating expert analogical reasoning;
22. Chulani & Boehm's *Bayesian tuning method* [6] for regression models allows an algorithm to carefully combine expert judgment with the available data;
23. This paper will argue for the use of heuristic *rejection rules* to represent expert intuitions on how to rank different effort models.

Fig. 1. Three different categories of effort estimation best practices: (top) expert-based, (middle) model-based, and (bottom) methods that combine expert and model-based.

COSEEKMO to select best parametric methods in the COCOMO format [3], [4], but it could just as easily be used to assess

- other model-based tools, like PRICE-S [17], SEER-SEM [18], or SLIM [19], and
- parametric versus nonlinear methods, e.g., a neural net or a genetic algorithm.

Biases in the data. Issues of sampling bias threaten any data mining experiment; i.e., what matters *there* may not be true *here*. For example, some of the data used here comes from NASA and NASA works in a particularly unique market niche. Nevertheless, we argue that results from NASA are relevant to the general software engineering industry. NASA makes extensive use of contractors. These contractors service many other industries. These contractors are contractually obliged (ISO-9001) to demonstrate their understanding and usage of current industrial best practices. For these reasons, noted researchers such as Basili et al. [20] have argued that conclusions from NASA data are relevant to the general software engineering industry.

	records	$T=$	average test error = MRE= $\frac{\lvert predicted_i - actual_i \rvert}{actual_i}$	
source:part	$\lvert train \rvert$	$\lvert test \rvert$	mean%	sd%
coc81:mode.org	13	10	32	27
coc81:lang.mol	10	10	34	29
coc81:all	53	10	42	45
coc81:mode.e	18	10	42	47
coc81:lang.ftn	14	10	50	48
coc81:kind.max	21	10	47	51
coc81:kind.mic	11	10	47	66
nasa93:cat.missionplan	10	10	46	45
nasa93:cat.avionicsmonitor	20	10	43	47
nasa93:fg.g	70	10	53	126
nasa93:project.X	28	10	68	142
nasa93:center.2	27	10	43	148
nasa93:mode.sd	59	10	58	149
nasa93:all	83	10	60	157
nasa93:project.Y	13	10	56	168
nasa93:center.5	29	10	80	169
nasa93:year.1975	27	10	82	192
nasa93:year.980	28	10	81	211
nasa93:mode.e	11	10	188	649

Fig. 2. Some effort modeling results, sorted by the standard deviation of the test error. Effort models were learned using Boehms's COCOMO-I "local calibration" procedure, described in Section 4.1 and Appendix D.

The data bias exists in another way: Our model-based methods use historical data and so are only useful in organizations that maintain this data on their projects. Such data collection is rare in organizations with low process maturity. However, it is common elsewhere, e.g., among government contractors whose contract descriptions include process auditing requirements. For example, it is common practice at NASA and the US Department of Defense to require a model-based estimate at each project milestone. Such models are used to generate estimates or to double-check an expert-based estimate.

Biases in the selection of data miners. Another source of bias in this study is the set of data miners explored by this study (linear regression, model trees, etc.). Data mining is a large and active field and any single study can use only a small subset of the known data mining algorithms. For example, this study does not explore the case-based reasoning methods favored by Shepperd and Schofield [9]. Pragmatically, it is not possible to explore all possible learners. The best we can do is to define our experimental procedure and hope that other researchers will apply it using a different set of learners. In order to encourage reproducibility, most of the data used in this study is available online.[1]

3 Background

3.1 COCOMO

The case study material for this paper uses COCOMO-format data. COCOMO (the COnstructive COst MOdel) was originally developed by Barry Boehm in 1981 [3] and was extensively revised in 2000 [4]. The core intuition behind COCOMO-based estimation is that, as a program grows in size, the development effort grows exponentially. More specifically,

$$effort(personmonths) = a * \left(KLOC^b\right) * \left(\prod_j EM_j\right). \quad (1)$$

1. See http://unbox.org/wisp/trunk/cocomo/data.

Mode	a	b	notes
Organic	3.2	1.05	projects from relatively small software teams develop software in a highly familiar, in-house environment.
Embedded	2.8	1.2	projects operating within (is embedded in) a strongly coupled complex of hardware, software, regulations, and operational procedures.
Semi-Detached	3.0	1.12	An intermediary mode between organic and embedded.

Fig. 3. Standard COCOMO 81 development modes.

Here, $KLOC$ is thousands of delivered source instructions. $KLOC$ can be estimated directly or via a *function point estimation*. Function points are a product of five defined data components (inputs, outputs, inquiries, files, external interfaces) and 14 weighted environment characteristics (data comm, performance, reusability, etc.) [4], [21]. A 1,000-line Cobol program would typically implement about 14 function points, while a 1,000-line C program would implement about seven.[2]

In (1), EM_j is an *effort multiplier* such as *cplx* (complexity) or *pcap* (programmer capability). In order to model the effects of EM_j on development effort, Boehm proposed reusing numeric values which he generated via regression on historical data for each value of EM_i (best practice #13 in Fig. 1).

In practice, effort data forms exponential distributions. Appendix B describes methods for using such distributions in effort modeling.

Note that, in COCOMO 81, Boehm identified three common types of software: *embedded*, *semidetached*, and *organic*. Each has their own characteristic "a" and "b" (see Fig. 3). COCOMO II ignores these distinctions. This study used data sets in both the COCOMO 81 and COCOMO II format. For more on the differences between COCOMO 81 and COCOMO II, see Appendix A.

3.2 Data

In this study, COSEEKMO built effort estimators using all or some *part* of data from three sources (see Fig. 4). *Coc*81 is the original COCOMO data used by Boehm to calibrate COCOMO 81. *CocII* is the proprietary COCOMO II data set. *Nasa*93 comes from a NASA-wide database recorded in the COCOMO 81 format. This data has been in the public domain for several years but few have been aware of it. It can now be found online in several places including the PROMISE (Predictor Models in Software Engineering) Web site.[3] *Nasa*93 was originally collected to create a NASA-tuned version of COCOMO, funded by the Space Station Freedom Program. *Nasa*93 contains data from six NASA centers, including the Jet Propulsion Laboratory. Hence, it covers a very wide range of software domains, development processes, languages, and complexity, as well as fundamental differences in culture and business practices between each center. All of these factors contribute to the large variances observed in this data set.

2. http://www.qsm.com/FPGearing.html.

3. http://promise.site.uottawa.ca/SERepository/ and http://unbox.org/wisp/trunk/cocomo/data.

When the *nasa*93 data was collected, it was required that there be multiple interviewers with one person leading the interview and one or two others recording and checking documentation. Each data point was cross-checked with either official records or via independent subjective inputs from other project personnel who fulfilled various roles on the project. After the data was translated into the COCOMO 81 format, the data was reviewed with those who originally provided the data. Once sufficient data existed, the data was analyzed to identify outliers and the data values were verified with the development teams once again if deemed necessary. This typically required from two to four trips to each NASA center. All of the supporting information was placed in binders, which we occasionally reference even today.

In summary, the large deviation seen in the *nasa*93 data of Fig. 2 is due to the wide variety of projects in that data set and *not* to poor data collection. Our belief is that *nasa*93 was collected using methods equal to or better than standard industrial practice. If so, then industrial data would suffer from deviations equal to or larger than those in Fig. 2.

3.3 Performance Measures

The performance of models generating continuous output can be assessed in many ways, including PRED(30), MMRE, correlation, etc. PRED(30) is a measure calculated from the relative error, or RE, which is the relative size of the difference between the actual and estimated value. One way to view these measures is to say that training data contains records with variables $1, 2, 3, \ldots, N$ and performance measures add additional new variables $N+1, N+2, \ldots$.

The magnitude of the relative error, or MRE, is the absolute value of that relative error:

$$MRE = |predicted - actual|/actual.$$

The mean magnitude of the relative error, or MMRE, is the average percentage of the absolute values of the relative errors over an entire data set. MMRE results are shown in Fig. 2 in the *mean% average test error* column. Given T tests, the MMRE is calculated as follows:

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

PRED(N) reports the average percentage of estimates that were within N percent of the actual values. Given T tests, then

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

For example, $\mathrm{PRED}(30) = 50\%$ means that half the estimates are within 30 percent of the actual.

Another performance measure of a model predicting numeric values is the correlation between predicted and actual values. Correlation ranges from $+1$ to -1 and a correlation of $+1$ means that there is a perfect positive linear relationship between variables. Appendix C shows how to calculate correlation.

All these performance measures (correlation, MMRE, and PRED) address subtly different issues. Overall, PRED

Data sources	*Coc81:*63 records in the COCOMO 81 format. Source: [3, p496-497]. Download from `http://unbox.org/wisp/trunk/cocomo/data/coc81modeTypeLangType.csv`. *Nasa93:* 93 NASA records in the COCOMO 81 format. Download from `http://unbox.org/wisp/trunk/cocomo/data/nasa93.csv`. *CocII:* 161 records in the COCOMO II format from the COCOMO consortium (co-ordinated by USC). This data is not in the public domain.
Data subsets	*All:* selects all records from a particular source; e.g. "coc81_all". *Category:* is a NASA-specific designation selecting the type of project; e.g. avionics, data capture, etc. *Dev:* indicates the development methodology; e.g. div.waterfall. *DevEnd:* shows the last year of the software project. *Fg:* selects either "*f*" (flight) or "*g*" (ground) software. *Kind:* selects records relating to the development platform; max= mainframe and mic= microprocessor. *Lang:* selects records about different development languages. *Project* and *center*: $nasa93$ designations selecting records relating to where the software was built and the name of the project. *Mode=e:* selects records relating to the *embedded* COCOMO 81 development mode. The different COCOMO 81 development models were described in Figure 3. *Mode=o:* selects COCOMO 81 *organic* mode records. *Mode=sd:* selects COCOMO 81 *semi-detached* mode records. *Org:* is a $cocII$ designation showing what organization provided the data. *Size:* is a $cocII$ specific designation grouping the records into (e.g.) those around 100KLOC. *Type:* selects different $coc81$ 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".

Fig. 4. Data sets (top) and parts (bottom) of the data used in this study.

measures how *well* an effort model performs, while MMRE measures *poor* performance. A single large mistake can skew the MMREs and not effect the PREDs. Shepperd and Schofield comment that

> MMRE is fairly conservative with a bias against over-estimates while PRED(30) will identify those prediction systems that are generally accurate but occasionally wildly inaccurate [9, p. 736].

Since they measure different aspects of model performance, COSEEKMO uses combinations of PRED, MMRE, and correlation (using the methods described later in this paper).

4 Designing COSEEKMO

When the data of Fig. 4 was used to train a COCOMO effort estimation model, the performance exhibited the large deviations seen in Fig. 2. This section explores sources and solutions of those deviations.

4.1 Not Enough Records for Training

Boehm et al. caution that learning on a small number of records is very sensitive to relatively small variations in the records used during training [4, p. 157]. A rule of thumb in regression analysis is that 5 to 10 records are required for every variable in the model [22]. COCOMO 81 has 15 variables. Therefore, according to this rule:

- Seventy-five to 150 records are needed for COCOMO 81 effort modeling.
- Fig. 2 showed so much variation in model performance because the models were built from too few records.

It is impractical to demand 75 to 150 records for training effort models. For example, in this study and in numerous other published works (e.g., [23], [4, p. 180], [10]), small training sets (i.e., tens, not hundreds, of records) are the norm for effort estimation. Most effort estimation data sets contain just a few dozen records or less.[4]

Boehm's solution to this problem is *local calibration* (hereafter, LC) [3, pp. 526–529]. LC reduces the COCOMO regression problem to just a regression over the two "*a*" and "*b*" constants of (1). Appendix D details the local calibration procedure.

In practice, LC is not enough to tame the large deviations problem. Fig. 2 was generated via LC. Note that, despite the restriction of the regression to just two variables, large deviations were still generated. Clearly, COSEEKMO needs to look beyond LC for a solution to the deviation problem.

4.2 Not Enough Records for Testing

The experiment results displayed in Fig. 2 had small test sets: just 10 records. Would larger test sets yield smaller deviations?

Rather than divide the project data into training and test sets, an alternative would be to train and test on all available records. This is *not* recommended practice. If the goal is to understand how well an effort model will work on future projects, it is best to assess the models via *holdout* records not used in training. Ferens and Christensen [23] report studies where the same project data was used to build effort models with 0 percent and 50 percent holdouts. A failure to use a holdout sample overstated a model's accuracy on new examples. The learned model had a PRED(30) of 57 percent over all the project data but a PRED(30) of only 28 percent on the holdout set.

The use of holdout sets is common practice [10], [24]. A standard holdout study divides the project data into a

4. Exceptions: In a personal communication, Donald Reifer reports that his company's databases contain thousands of records. However, this information in these larger databases is proprietary and generally inaccessible.

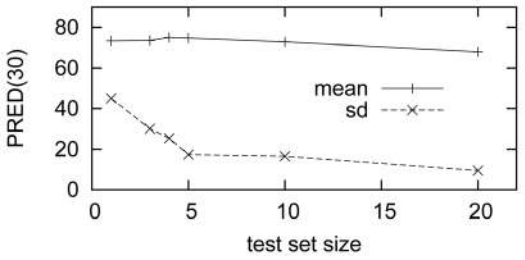

Fig. 5. Effects of test set size on PRED(30) for *nasa*93.

66 percent training set and a 33 percent test set. Our project data sets ranged in size from 20 to 200 and, so, 33 percent divisions would generate test sets ranging from 7 to 60 records. Lest this introduced a conflating factor to this study, we instead used test sets of fixed size.

These fixed test sets must be as small as possible. Managers assess effort estimators via their efficacy in some local situation. If a model fails to produce accurate estimates, then it is soon discarded. The following principle seems appropriate:

> *Effort estimators should be assessed via small test sets since that is how they will be assessed in practice.*

Test sets of size 10 were chosen after conducting the experiment of Fig. 5. In that figure, 30 times, X records were selected at random to be the holdout set and LC was applied to the remaining records. As X increased, the standard deviation of the PRED(30) decreased (and most of that decrease occurred in the range $X = 1$ to $X = 5$). Some further decrease was seen up to $X = 20$, but, in adherence to the above principle (and after considering certain prior results [10]), COSEEKMO used a fixed test set of size 10.

4.3 Wrong Assumptions

Every modeling framework makes assumptions. The wrong assumptions can lead to large deviation between predicted and actual values (such as those seen in Fig. 2) when the wrong equations are being forced to fit the project data.

In the case of COCOMO, those assumptions come in two forms: the constants and the equations that use the constants. For example, local calibration assumes that the values for the scale factors and effort multipliers are correct and we need only to adjust "a" and "b." In order to check this assumption, COSEEKMO builds models using both the precise values *and* the simpler proximal values (the precise unrounded values and the proximal values are shown in Appendix E).

Another COCOMO assumption is that the development effort conforms to the effort equations of (1) (perhaps linearized as described in Appendix B). Many regression methods make this linear assumption, i.e., they fit the project data to a single straight line. The line offers a set of predicted values; the distance from these predicted values to the actual values is a measure of the error associated with that line. Linear regression tools, such as the least squares regression package used below (hereafter, LSR), search for lines that minimize that sum of the squares of the error.

Linearity is not appropriate for all distributions. While some nonlinear distributions can be transformed into linear functions, others cannot. A linear assumption might suffice

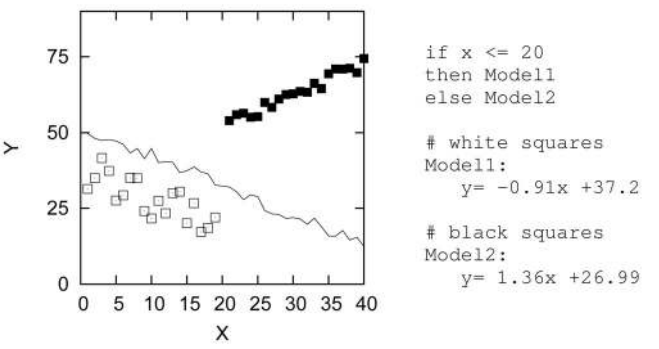

Fig. 6. Linear and nonlinear distributions shown as a line and squares (respectively).

for the line shown in the middle of Fig. 6. However, for the square points, something radically alters the "$Y = f(X)$" around $X = 20$. Fitting a single linear function to the white *and* black square points would result in a poorly fitting model.

A common method for handling arbitrary distributions to approximate complex distributions is via a set of piecewise linear models. *Model tree learners*, such as Quinlan's M5P algorithm [25], can learn such piecewise linear models. M5P also generates a decision tree describing when to use which linear model. For example, M5P could represent the squares in Fig. 6 as two linear models in the model tree shown on the right of that figure.

Accordingly, COSEEKMO builds models via LC (local calibration) and LSR (least squares linear regression), as well as M5P (model trees), using either the precise (unrounded) or proximal COCOMO numerics.

4.4 Model Too Big

Another way to reduce deviations is to reduce the number of variables in a model. Miller makes a compelling argument for such pruning: Decreasing the number of variables decreases the deviation of a linear model learned by minimizing least squares error [14]. That is, the fewer the columns, the more restrained the model predictions. In results consistent with Miller's theoretical results, Kirsopp and Shepperd [15] and Chen et al. [12] report that variable pruning improves effort estimation.

COSEEKMO's variable pruning method is called the "WRAPPER" [26]. The WRAPPER passes different subsets of variables to some oracle (in our case, LC/LSR/M5P) and returns the subset which yields best performance (for more details on the WRAPPER, see Appendix F). The 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 project data sets in this study are small enough to permit the use of the WRAPPER.

4.5 Noise and Multiple-Correlations

Learning an effort estimation model is easier when the learner does not have to struggle with fitting the model to confusing "noisy" project data (i.e., when the project data contains spurious signals not associated with variations to projects). Noise can come from many sources such as clerical errors or missing variable values. For example,

organizations that only build word processors may have little project data on software requiring high reliability.

On the other hand, if two variables are tightly correlated, then using both diminishes the likelihood that either will attain significance. A repeated result in data mining is that pruning some correlated variables increases the effectiveness of the learned model (the reasons for this are subtle and vary according to which particular learner is being used [27]).

COSEEKMO handles noise and multiple correlations via the WRAPPER. Adding a noisy or multiply correlated variable would not improve the performance of the effort estimator, so the WRAPPER would ignore them.

4.6 Algorithm

COSEEKMO explores the various effort estimation methods described above. Line 03 of COSEEKMO skips all subsets of the project data with less than 20 records (10 for training, 10 for testing). If not skipped, then line 08 converts symbols like "high" or "very low" to numerics using the precise *or* proximal cost drivers shown in Appendix E.

The **print** statements from each part are grouped by the experimental $treatment$, i.e., some combination of

$$treatment =< Datum, Numbers, Learn, Subset >:$$

- line 01 picks the *Datum*,
- line 08 picks the *Numbers*,
- line 21 picks the *Learn*er, and
- line 18 picks the size of the variables *Subset* ($|Subset|$).

For the COCOMO 81 project data sets, if $|Subset| = 17$, then the learner used the actual effort, lines of code, and all 15 effort multipliers. Smaller subsets (i.e., $|Subset| < 17$) indicate COCOMO 81 treatments where the WRAPPER reported that some subset of the variables might be worth exploring. For the COCOMO II project data sets, the analogous thresholds are $|Subset| = 24$ and $|Subset| < 24$.

In *Experimental Methods for Artificial Intelligence*, Cohen advises comparing the performance of a supposedly more sophisticated approach against a simpler "straw man" method [28, p. 81]. The rules of Fig. 7 hence include "straw man" tests. First, COSEEKMO is superfluous if estimates learned from just lines of code perform as well as any other method. Hence, at line 17, we ensure that one of the variable subsets explored is just lines of code and effort. Results from this "straw man" are recognizable when the variable subset size is 2; i.e., $|Subset| = 2$.

Secondly, COSEEKMO is superfluous if it is outperformed by COCOMO 81. To check this, off-the-shelf COCOMO 81 (i.e., (1)) is applied to the COCOMO 81 project data, assuming that the software is an embedded system (line 12), a semidetached system (line 13), or an organic (line 14) system. For these assumptions, (1) is applied using the appropriate "a" and "b" values taken from Fig. 3. In order to distinguish these results from the rest, they are labeled with a *Learn* set as "e," sd," or "org" for "embedded," "semidetached," or "organic" (this second "straw man" test was omitted for COCOMO II since "embedded, semidetached, organic" are only COCOMO 81 concepts).

```
01 for Datum ∈ {nasa93,coc81,cocII}
02   for Part ∈ Parts
03     if Data.Part.size ≥ 20
04     then
05       30 times do
06         Test0 ← Data.Part.any(10)    # note: random selection
07         Train0 ← Data.Part - Test0
08         for Numbers ∈ precise or proximal
09           Test1 ← Test0.with(Numbers)
10           Train1 ← Train0.with(Numbers)
11           if ( Data = coc81 or Data = nasa93 )
12           then print estimates of Test1 using a←2.8 and b←1.2
13                print estimates of Test1 using a←3.0 and b←1.12
14                print estimates of Test1 using a←3.2 and b←1.05
15           fi
16           Subsets ← WRAPPER(Train1)
17           Subsets ← Subsets + {KLOC,effort}
18           for Subset ∈ Subsets
19             Test2 ← Test1.variables(Subset)
20             Train2 ← Train1.variables(Subset)
21             for Learn ∈ {LC, LSR, M5P}
22               Model ← Learn(Train2)
23               print estimates of Test2 using Model
```

Fig. 7. COSEEKMO: generating models. For a definition of the project data parts referred to on line 2, see Fig. 4.

4.7 Rejection Rules

COSEEKMO collects the data generated by the **print** statements of Fig. 7's rejection rules and sorts those results by *treatment*. The values of different treatments are then assessed via a set of *rejection rules*.

The rejection rules act like contestants in a competition where "losers" must be ejected from the group. Treatments are examined in pairs and, each time the rules are applied, one member of the pair is ejected. This process repeats until none of the rules can find fault with any of the remaining treatments. When the rules stop firing, all the *surviving* treatments are printed.

The five rules in our current system appear in the worse function of Fig. 8:

- *Rule1* is a statistical test condoned by standard statistical textbooks. If a two-tailed t-test reports that the means of two treatments are statistically different, then we can check if one mean is less than the other.
- *Rule2* checks for correlation, i.e., what treatments track best between predicted and actual values.
- *Rule3* checks for the size of the standard deviation and is discussed below.
- *Rule4* was added because PRED is a widely used performance measure for effort models [6].
- *Rule5* rejects treatments that have similar performance but use more variables.

Since these rules are the core of COSEEKMO, we wrote them with great care. One requirement that we had was that the rules could reproduce at least one historical expert effort estimation study. As discussed below, the current set of rejection rules can reproduce Boehm's 1981 COCOMO 81 analysis (while an earlier version of the rules favored treatments that were radically different from those used by Boehm).

The ordering of tests in Fig. 8's worse function imposes a rule priority (lower rules can fire only if higher rules fail).

```
function reject(treatments)
   survivors = prune(treatments)
   print survivors

function prune(treatments)
   for x ∈ treatments
      for y ∈ treatments
         if x ≠ y
            if (z=worse(x,y)) then return prune1(z,treatments) fi
   return treatments  # if nothing found to prune

function prune1(z,treatments)
   delete z from treatments
   return prune(treatments)

function worse(x,y)
   if    statisticallyDifferent(x,y)
   then
         if error(x) < error(y)                    then return y fi  # rule1
         if error(y) < error(x)                    then return x fi  # rule1
   else
         if correlation(x) < correlation(y)         then return x fi  # rule2
         if correlation(y) < correlation(x)         then return y fi  # rule2

         if sd(x)/mean(x)< sd(y)/mean(y)           then return y fi  # rule3
         if sd(y)/mean(y)< sd(x)/mean(x)           then return x fi  # rule3

         if pred(x)       < pred(y)                then return x fi  # rule4
         if pred(y)       < pred(x)                then return y fi  # rule4

         if |Subset(x)|   < |Subset(y)|            then return y fi  # rule5
         if |Subset(y)|   < |Subset(x)|            then return x fi  # rule5
   fi
   return 0  # if no reason to return true
```

Fig. 8. COSEEKMO's current rejection rules. *Error* refers to the MMRE. *Correlation* refers to the connection of the expected to actual effort (see Appendix C). *Worse*'s statistically difference test compares two MMREs x and y using a two-tailed t-test at the 95 percent confidence interval; i.e., $\frac{|mean(x)-mean(y)|}{\sqrt{(sd(x)^2/(n(x)-1))+sd(y)^2/(n(y)-1)}} > 1.96$.

Well-founded statistical tests are given higher priority than heuristic tests. Hence, *rule1* is listed first and *rule4* and *rule5* are last. *Rule2* was made higher priority than *rule3* since that prioritization could still reproduce Boehm's 1981 result for embedded and organic systems.

This prioritization influences how frequently the different rules are applied. As shown in Fig. 9, lower priority rules (e.g., *rule5*) fire far less frequently that higher priority rules (e.g., *rule1*) since the lower priority rules are only tested when all the higher priority rules have failed.

5 Experiments

Fig. 10 shows the survivors after running the rejection rules of Fig. 8 on those parts of *coc81*, *nasa93*, and *cocII* with 20 or more records. Several aspects of that figure are noteworthy. First, COSEEKMO's rejection rules were adjusted till 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 project data. For example:

- The embedded and organic "a" and "b" values worked best for *coc81* embedded and organic systems (rows 4 and 5).

	checks for..	COCOMO 81 (nasa93, coc81)	COCOMO II (cocii)
rule1	mmre	59.2%	63.6%
rule2	correlation	34.5%	28.7%
rule3	sd/mmre	5.5%	3.6%
rule4	pred	0.1%	3.0%
rule5	subset size	0.7%	1.0%

Fig. 9. Percent frequency of rule firings.

- Local calibration was called only once for *coc81.all* (row 6) 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, semidetached, or organic systems.

Since COSEEKMO's rules were tuned to reproduce Boehm's COCOMO 81 analysis, it is hardly surprising that the COCOMO II results also use Boehm techniques: lines 20 to 28 of Fig. 10 all use Boehm's preferred local calibration (LC) method.

Second, COSEEKMO can reduce *both* the standard deviation *and* the mean estimate error. For example, COSEEKMO reduces the {min, median, max} MRE mean results from {43, 58, 188} percent (in Fig. 2) to {22, 38, 64} percent (in Fig. 10). But the reductions in standard deviation are far larger and, given the concerns of this paper, far more significant. Fig. 2 showed that Boehm's local calibration method yielded models from *nasa93* with {min, median, max} MRE standard deviations of {45,157,649} percent (respectively). However, using COSEEKMO, Fig. 10 shows that the *nasa93* {min, median, max} MRE standard deviations were {20, 39, 100} percent.

Third, in 14 cases (rows 8, 9, 12, 13, 15, 16, 17, 18, 21, 22, 23, 26, 27, and 28) the WRAPPER discarded, on average, five to six variables and sometimes many more (e.g., in rows 9, 13, 22, and 27, the surviving models discarded nearly half the variables). This result, together with 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 [3], [4]. 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.

Fourth, many of the results in Fig. 10 use nonstandard effort estimation methods. As mentioned above, in 14 experiments, the WRAPPER was useful; i.e., some of the standard COCOMO attributes were ignored. Also, in four cases, the best effort models were generated using linear regression (rows 12, 15, and 16) or model trees (row 13). Further, in six cases (rows 9, 10, 12, 16, 22, and 28), COSEEKMO found that the precise COCOMO numerics were superfluous and that the proximal values sufficed. That is, reusing regression parameters learned from prior projects was not useful. More generally, standard effort estimation methods are not optimal in all cases.

Fifth, even though COSEEKMO has greatly reduced the variance in NASA project data, the deviations for the NASA data are much larger than with COCOMO. As discussed in Section 3.2, the *nasa93* data set comes from across the NASA

		Records		Treatment			Results		
	:						mean	MRE	
row	source:part	$t = \|train\|$	$T = \|test\|$	Numbers	$\|Subset\|$	Learn	PRED(30)	mean%	sd%
1.	coc81:kind.mic	11	10	precise	17	e	60	31	21
2.	coc81:lang.ftn	14	10	precise	17	sd	42	44	30
3.	coc81:kind.max	21	10	precise	17	e	52	38	33
4.	coc81:mode.org	13	10	precise	17	org	62	32	33
5.	coc81:mode.e	18	10	precise	17	e	46	40	34
6.	coc81:all	53	10	precise	17	LC	50	40	37
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:mode.sd	59	10	proximal	7	LC	62	33	34
10.	nasa93:category.missionplanning	10	10	proximal	17	e	50	36	37
11.	nasa93:center.2	27	10	precise	17	LC	83	22	38
12.	nasa93:fg.g	70	10	proximal	10	LSR	65	32	39
13.	nasa93:category.avionicsmonitoring	20	10	precise	8	M5P	53	38	39
14.	nasa93:project.X	28	10	precise	17	e	42	42	45
15.	nasa93:year.1975	27	10	precise	11	LSR	52	50	62
16.	nasa93:all	83	10	proximal	14	LSR	43	48	62
17.	nasa93:center.5	29	10	precise	12	LC	43	57	70
18.	nasa93:year.1980	28	10	precise	16	LC	53	53	80
19.	nasa93:mode.e	11	10	precise	17	e	42	64	100
20.	cocII:org.1	24	10	precise	24	LC	100	8	5
21.	cocII:devEnd.1998	44	10	precise	22	LC	95	11	9
22.	cocII:lang.c	11	10	proximal	11	LC	79	19	13
23.	cocII:dev.waterfall	38	10	precise	23	LC	64	23	18
24.	cocII:org.2	38	10	precise	24	LC	74	20	18
25.	cocII:devEnd.1980	38	10	precise	24	LC	72	19	18
26.	cocII:all	151	10	precise	21	LC	76	21	21
27.	cocII:devEnd.1991	12	10	precise	11	LC	51	34	25
28.	cocII:devEnd.1996	28	10	proximal	17	LC	47	40	29

Fig. 10. Survivors from *Rejection Rules 1, 2, 3, 4, and 5*. In the column labeled "Learn," the rows *e*, *sd*, and *org* denote cases where using COCOMO 81 with the *embedded*, *semidetached*, *organic* effort multipliers of Fig. 3 proved to be the best treatment.

enterprise and, hence, is very diverse (different tools, domains, development cultures, and business processes). Consequently, *nasa93* suffers greatly from large deviations.

Sixth, looking across the rows, there is tremendous variation in what treatment proved to the "best" treatment. This pattern appears in the general data mining literature: Different summarization methods work best in different situations [29] and it remains unclear why that is so. This variation in the "best" learner method is particularly pronounced in small data sets (e.g., our NASA and COCOMO data), where minor quirks in the data can greatly influence learner performance. However, we can speculate why methods *Learn*ers other than LC predominate for non-COCOMO data such as *nasa93*. Perhaps fiddling with two tuning parameters may be appropriate when the variance in the data is small (e.g., the *cocII* results that all used LC) but can be inappropriate when the variance is large (e.g., in the *nasa93* data).

Last, and reassuringly, the "straw man" result of $|Subset| = 2$ never appeared. That is, for all parts of our project data, COSEEKMO-style estimations were better than using just lines of code.

6 Applications of COSEEKMO

Applications of COSEEKMO use the tool in two subtly different ways. Fig. 10 shows the surviving treatments after applying the rejection rules *within* particular project data sets. Once the survivors from different data sets are generated, the rejection rules can then be applied *across* data sets; i.e., by comparing different rows in Fig. 10 (note that such *across* studies use *within* studies as a preprocessor).

Three COSEEKMO applications are listed below:

- *Building effort models* uses a *within* study like the one that generated Fig. 10.
- On the other hand, *assessing data sources* and *validating stratifications* use *across* studies.

6.1 Building Effort Models

Each line of Fig. 10 describes a best treatment (i.e., some combination of $\langle Datum, Numbers, Learn, Subset \rangle$) for a particular data set. Not shown are the random number seeds saved by COSEEKMO (these seeds control how COSEEKMO searched through its data).

Those recommendations can be imposed as constraints on the COSEEKMO code of Fig. 7 to restrict, e.g., what techniques are selected for generating effort models. With those constraints in place, and by setting the random number seed to the saved value, COSEEKMO can be re-executed using the best treatments to produce 30 effort estimation models (one for each repetition of line 5 of Fig. 7). The average effort estimation generated from this ensemble could be used to predict the development effort of a particular project.

6.2 Assessing Data Sources

Effort models are generated from project databases. Such databases are assembled from multiple sources. Experienced effort modelers know that some sources are more trustworthy than others. COSEEKMO can be used to test if a new data source would improve a database of past project data.

To check the merits of adding a new data source Y to a existing data of past project data X, we would:

- First run the code of Fig. 7 *within* two data sets: 1) X and 2) $X + Y$.
- Next, the rejection rules would be executed *across* the survivors from X and $X + Y$. The new data source Y

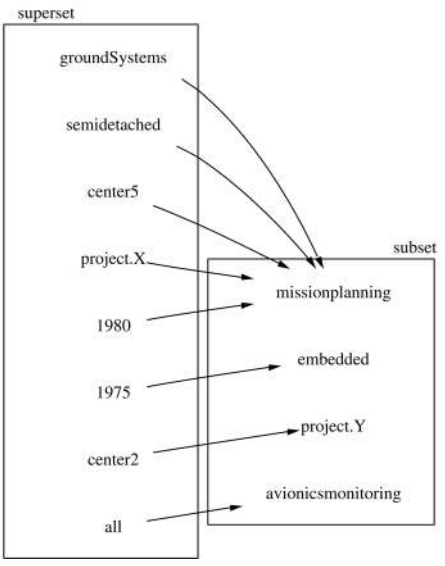

Fig. 11. Supersets of *nasa* rejected in favor of subset stratifications.

should be added to X if $X + Y$ yields a better effort model than X (i.e., is not culled by the rejection rules).

6.3 Validating Stratifications

A common technique for improving effort models is the *stratification* of the *superset* of all data into *subsets* of related data [1], [4], [9], [10], [23], [30], [31], [32], [33]. Subsets containing related projects have less variation and so can be easier to calibrate. Various experiments demonstrate the utility of stratification [4], [9], [10].

Databases of past project data contain hundreds of candidate stratifications. COSEEKMO *across* studies can assess which of these stratifications actually improve effort estimation. Specifically, a stratification subset is "better" than a superset if, in an *across* data set study, the rejection rules cull the superset. For example, *nasa93* has the following subsets:

- all the records,
- whether or not it is *flight* or *ground* system,
- the *parent project* that includes the project expressed in this record,
- the NASA center where the work was conducted,
- etc.

Some of these subsets are bigger than others. Given a record labeled with subsets $\{S_1, S_2, S_3, \ldots\}$ then we say a *candidate stratification* has several properties:

- It contains records labeled $\{S_i, S_j\}$.
- The number of records in S_j is greater than S_i; i.e., S_j is the superset and S_i is the stratification.
- S_j contains 150 percent (or more) the number of records in S_i.
- S_i contains at least 20 records.

Nasa93 contains 207 candidate stratifications (including one stratification containing all the records). An *across study* showed that only the four subsets S_i shown in Fig. 11 were "better" than their large supersets S_j. That is, while stratification *may* be useful, it should be used with caution since it does not always improve effort estimation.

7 Conclusion

Unless it is tamed, the large deviation problem prevents the effort estimation community from comparatively assessing the merits of different supposedly best practices. COSEEKMO was designed after an analysis of several possible causes of these large deviations. COSEEKMO can comparatively assess different model-based estimation methods since it uses *more* than just standard parametric t-tests (correlation, number of variables in the learned model, etc.).

The *nasa93* results from Fig. 2 and Fig. 10 illustrate the "before" and "after" effects of COSEEKMO. Before, using Boehm's local calibration method, the effort model MRE errors were {43,58,188} percent for {min,median,max} (respectively). After, using COSEEKMO, those errors had dropped to {22, 38, 64} percent. Better yet, the MRE standard deviation dropped from {45, 157, 649} percent to {20, 39, 100} percent. Such large reductions in the deviations increases the confidence of an effort estimate since they imply that an assumption about a particular point estimate is less prone to inaccuracies.

One advantage of COSEEKMO is that the analysis is fully automatic. The results of this paper took 24 hours to process on a standard desktop computer; that is, it is practical to explore a large number of alternate methods within the space of one weekend. On the other hand, COSEEKMO has certain restrictions, e.g., input project data must be specified in a COCOMO 81 or COCOMO-II format.[5] However, the online documentation for COCOMO is extensive and our experience has been that it is a relatively simple matter for organizations to learn COCOMO and report their projects in that format.

A surprising result from this study was that many of the best effort models selected by COSEEKMO were not generated via techniques widely claimed to be "best practice" in the model-based effort modeling literature. For example, recalling Fig. 1, the study above shows numerous examples where the following supposedly "best practices" were outperformed by other methods:

- *Reusing old regression parameters* (Fig. 1, number 13). In Fig. 10, the *precise* COCOMO parameters sometimes were outperformed by the proximal parameters.
- *Stratification* (Fig. 1, number 16). Only four (out of 207) candidate stratifications in $Nasa93$ demonstrably improved effort estimation.
- *Local calibration* (Fig. 1, number 17). Local calibration (LC) was often not the best treatment seen in the $Learn$ column of Fig. 10.

Consequently, we advise that 1) any supposed "best practice" in model-based effort estimation should be viewed as a *candidate* technique which *may or may not* be useful in a particular domain, and 2) tools like COSEEKMO should be used to help analysts explore and select the best practices for their particular domain.

5. For example, http://sunset.usc.edu/research/COCOMOII/expert_cocomo/drivers.html.

8 Future Work

Our next step is clear. Having tamed large deviations in model-based methods, it should now be possible to compare model-based and expert-based approaches.

Also, it would be useful see how COSEEKMO behaves on data sets with less than 20 records.

Further, there is much growing literature on combining the results from multiple learners (e.g., [34], [35], [36], and [37]). In noisy or uncertain environments (which seems to characterize the effort estimation problem), combining the conclusions from *committees* of automatically generated experts might perform better than just relying on a single expert. This is an exciting option which needs to be explored.

Appendix A
COCOMO I versus COCOMO II

In COCOMO II, the exponential COCOMO 81 term b was expanded into the following expression:

$$b + 0.01 * \sum_j SF_j, \tag{2}$$

where b is 0.91 in COCOMO II 2000, and SF_j is one of five *scale factors* that exponentially influence effort. Other changes in COCOMO II included dropping the development modes of Fig. 3 as well as some modifications to the list of effort multipliers and their associated numeric constants (see Appendix E).

Appendix B
Logarithmic Modeling

COCOMO models are often built via linear least squares regression. To simplify that process, it is common to transform a COCOMO model into a linear form by taking the natural logarithm of (1):

$$ln(effort) = ln(a) + b * ln(KLOC) + ln(EM_1) + \ldots . \tag{3}$$

This linear form can handle COCOMO 81 and COCOMO II. The scale factors of COCOMO II affect the final effort exponentially according to KLOC. Prior to applying (3) to COCOMO II data, the scale factors SF_j can be replaced with

$$SF_j = 0.01 * SF_j * ln(KLOC). \tag{4}$$

If (3) is used, then *before* assessing the performance of a model, the estimated effort has to be converted back from a logarithm.

Appendix C
Calculating Correlation

Given a test set of size T, correlation is calculated as follows:

$$\bar{p} = \frac{\sum_I^T predicted_i}{T}, \qquad \bar{a} = \frac{\sum_I^T actual_i}{T},$$

$$S_p = \frac{\sum_i^T (predicted_i - \bar{p})^2}{T-1}, \qquad S_a = \frac{\sum_i^T (actual_i - \bar{a})^2}{T-1},$$

$$S_{pa} = \frac{\sum_i^T (predicted_i - \bar{p})(actual_i - \bar{a})}{T-1},$$

$$corr = S_{pa} / \sqrt{S_p * S_a}.$$

upper: increase these to decrease effort	acap: analysts capability
	pcap: programmers capability
	aexp: application experience
	modp: modern programming practices
	tool: use of software tools
	vexp: virtual machine experience
	lexp: language experience
middle	sced: schedule constraint
lower: decrease these to increase effort	data: data base size
	turn: turnaround time
	virt: machine volatility
	stor: main memory constraint
	time: time constraint for cpu
	rely: required software reliability
	cplx: process complexity

Fig. 12. COCOMO 81 effort multipliers.

Appendix D
Local Calibration

This approach assumes that a matrix $D_{i,j}$ holds

- the natural log of the $KLOC$ estimates,
- the natural log of the actual efforts for projects $i \leq j \leq t$, and
- the natural logarithm of the cost drivers (the scale factors and effort multipliers) at locations $1 \leq i \leq 15$ (for COCOMO 81) or $1 \leq i \leq 22$ (for COCOMO-II).

With those assumptions, Boehm [3] shows that, for COCOMO 81, the following calculation yields estimates for "a" and "b" that minimize the sum of the squares of residual errors:

$$\left.\begin{array}{rcl} EAF_i & = & \sum_j^N D_{i,j}, \\ a_0 & = & t, \\ a_1 & = & \sum_i^t KLOC_i, \\ a_2 & = & \sum_i^t (KLOC_i)^2, \\ d_0 & = & \sum_i^t (actual_i - EAF_i), \\ d_1 & = & \sum_i^t ((actual_i - EAF_i) * KLOC_i), \\ b & = & (a_0 d_1 - a_1 * d_0)/(a_0 a_2 - a_1^2), \\ a_3 & = & (a_2 d_0 - a_1 d_1)/(a_0 a_2 - a_1^2), \\ a & = & e^{a_3}. \end{array}\right\} \tag{5}$$

Appendix E
COCOMO Numerics

Fig. 12 shows the COCOMO 81 EM_j (effort multipliers). The effects of those multipliers on the effort are shown in Fig. 13. Increasing the *upper* and *lower* groups of variables will *decrease* or *increase* the effort estimate, respectively.

Fig. 14 shows the COCOMO 81 effort multipliers of Fig. 13, proximal and simplified to two significant figures.

Fig. 15, Fig. 16, and Fig. 17 show the COCOMO-II values analogies to Fig. 12, Fig. 13, and Fig. 14 (respectively).

Appendix F
The WRAPPER

Starting with the empty set, the WRAPPER adds some combinations of columns 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 variables and checks if a better model

		very low	low	nominal	high	very high	extra high
upper (increase these to decrease effort)	ACAP	1.46	1.19	1.00	0.86	0.71	
	PCAP	1.42	1.17	1.00	0.86	0.70	
	AEXP	1.29	1.13	1.00	0.91	0.82	
	MODP	1.2	1.10	1.00	0.91	0.82	
	TOOL	1.24	1.10	1.00	0.91	0.83	
	VEXP	1.21	1.10	1.00	0.90		
	LEXP	1.14	1.07	1.00	0.95		
middle	SCED	1.23	1.08	1.00	1.04	1.10	
lower (increase these to increase effort)	DATA		0.94	1.00	1.08	1.16	
	TURN		0.87	1.00	1.07	1.15	
	VIRT		0.87	1.00	1.15	1.30	
	STOR			1.00	1.06	1.21	1.56
	TIME			1.00	1.11	1.30	1.66
	RELY	0.75	0.88	1.00	1.15	1.40	
	CPLX	0.70	0.85	1.00	1.15	1.30	1.65

Fig. 13. The precise COCOMO 81 effort multiplier values.

		very low	low	nominal	high	very high	extra high
upper (increase these to decrease effort)	ACAP	1.2	1.1	1.00	0.9	0.8	
	PCAP	1.2	1.1	1.00	0.9	0.8	
	AEXP	1.2	1.1	1.00	0.9	0.8	
	MODP	1.2	1.1	1.00	0.9	0.8	
	TOOL	1.2	1.1	1.00	0.9	0.8	
	VEXP	1.2	1.1	1.00	0.9		
	LEXP	1.2	1.1	1.00	0.9		
middle	SCED	1.2	1.1	1.00	1.1	1.2	
lower (increase these to increase effort)	DATA		0.9	1.00	1.1	1.2	
	TURN		0.9	1.00	1.1	1.2	
	VIRT		0.9	1.00	1.1	1.2	
	STOR			1.00	1.1	1.2	1.3
	TIME			1.00	1.1	1.2	1.3
	RELY	0.8	0.9	1.00	1.1	1.2	
	CPLX	0.8	0.9	1.00	1.1	1.2	1.3

Fig. 14. Proximal COCOMO 81 effort multiplier values.

comes from learning over the larger set of variables. The WRAPPER stops when there are no more variables 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./ip1>COSEEKMO uses the WRAPPER since experiments by other researchers strongly suggest that it is superior to many other variable pruning methods. For example, Hall and Holmes [27] compare the WRAPPER to several other variable pruning methods including principal component

scale factors (exponentially decrease effort)	prec: have we done this before?
	flex: development flexibility
	resl: any risk resolution activities?
	team: team cohesion
	pmat: process maturity
upper (linearly decrease effort)	acap: analyst capability
	pcap: programmer capability
	pcon: programmer continuity
	aexp: analyst experience
	pexp: programmer experience
	ltex: language and tool experience
	tool: tool use
	site: multiple site development
	sced: length of schedule
lower (linearly increase effort)	rely: required reliability
	data: secondary memory storage requirements
	cplx: program complexity
	ruse: software reuse
	docu: documentation requirements
	time: runtime pressure
	stor: main memory requirements
	pvol: platform volatility

Fig. 15. The COCOMO II scale factors and effort multipliers.

		extra low	very low	low	nominal	high	very high	extra high
scale factors (exponentially decreases effort)	prec		6.20	4.96	3.72	2.48	1.24	0.00
	flex		5.07	4.05	3.04	2.03	1.01	0.00
	resl		7.07	5.65	4.24	2.83	1.41	0.00
	team		5.48	4.38	3.29	2.19	1.10	0.00
	pmat		7.80	6.24	4.68	3.12	1.56	0.00
upper (linearly decreases effort)	acap		1.42	1.19	1.00	0.85	0.71	
	pcap		1.34	1.15	1.00	0.88	0.76	
	pcon		1.29	1.12	1.00	0.90	0.81	
	aexp		1.22	1.10	1.00	0.88	0.81	
	pexp		1.19	1.09	1.00	0.91	0.85	
	ltex		1.20	1.09	1.00	0.91	0.84	
	tool		1.17	1.09	1.00	0.90	0.78	
	site		1.22	1.09	1.00	0.93	0.86	0.80
	sced		1.43	1.14	1.00	1.00	1.00	
lower (linearly increases effort)	rely		0.82	0.92	1.00	1.10	1.26	
	data			0.90	1.00	1.14	1.28	
	cplx		0.73	0.87	1.00	1.17	1.34	1.74
	ruse			0.95	1.00	1.07	1.15	1.24
	docu		0.81	0.91	1.00	1.11	1.23	
	time				1.00	1.11	1.29	1.63
	stor				1.00	1.05	1.17	1.46
	pvol			0.87	1.00	1.15	1.30	

Fig. 16. The precise COCOMO II numerics.

analysis (PCA—a widely used technique). Column pruning methods can be grouped according to

- whether or not they make special use of the target variable in the data set, such as "development cost" and
- whether or not pruning uses the target learner.

PCA is unique since it *does not* make special use of the target variable. The 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 variable), while the WRAPPER was the best (since it can exploit its special knowledge of the target learner).

ACKNOWLEDGMENTS

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology,

		extra low	very low	low	nominal	high	very high	extra high
Scale Factors	PREC		6.3	5.1	3.8	2.5	1.3	0
	FLEX		6.3	5.1	3.8	2.5	1.3	0
	RESL		6.3	5.1	3.8	2.5	1.3	0
	TEAM		6.3	5.1	3.8	2.5	1.3	0
	PMAT		6.3	5.1	3.8	2.5	1.3	0
upper	ACAP		1.3	1.1	1.0	0.9	0.8	
	PCAP		1.3	1.1	1.0	0.9	0.8	
	PCON		1.3	1.1	1.0	0.9	0.8	
	AEXP		1.3	1.1	1.0	0.9	0.8	
	PEXP		1.3	1.1	1.0	0.9	0.8	
	LTEX		1.3	1.1	1.0	0.9	0.8	
	TOOL		1.3	1.1	1.0	0.9	0.8	
	SITE		1.3	1.1	1.0	0.9	0.8	0.8
	SCED		1.3	1.1	1.0	0.9	0.8	
lower	RELY		0.8	0.9	1.0	1.1	1.3	
	DATA			0.9	1.0	1.1	1.3	
	CPLX		0.8	0.9	1.0	1.1	1.3	1.5
	RUSE			0.9	1.0	1.1	1.3	1.5
	DOCU		0.8	0.9	1.0	1.1	1.3	
	TIME				1.0	1.1	1.3	1.5
	STOR				1.0	1.1	1.3	1.5
	PVOL			0.9	1.0	1.1	1.3	

Fig. 17. The proximal COCOMO II numerics.

under a contract with the US 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 US Government. See http://menzies.us/pdf/06coseekmo.pdf for an earlier draft of this paper.

REFERENCES

[1] K. Lum, J. Powell, and J. Hihn, "Validation of Spacecraft Cost Estimation Models for Flight and Ground Systems," *Proc. Conf. Int'l Soc. Parametric Analysts (ISPA), Software Modeling Track,* May 2002.

[2] M. Jorgensen, "A Review of Studies on Expert Estimation of Software Development Effort," *J. Systems and Software,* vol. 70, nos. 1-2, pp. 37-60, 2004.

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

[4] 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.

[5] B.B.S. Chulani, B. Clark, and B. Steece, "Calibration Approach and Results of the Cocomo II Post-Architecture Model," *Proc. Conf. Int'l Soc. Parametric Analysts (ISPA),* 1998.

[6] S. Chulani, B. Boehm, and B. Steece, "Bayesian Analysis of Empirical Software Engineering Cost Models," *IEEE Trans. Software Eng.,* vol. 25, no. 4, July/Aug. 1999.

[7] C. Kemerer, "An Empirical Validation of Software Cost Estimation Models," *Comm. ACM,* vol. 30, no. 5, pp. 416-429, May 1987.

[8] R. Strutzke, *Estimating Software-Intensive Systems: Products, Projects and Processes.* Addison Wesley, 2005.

[9] M. Shepperd and C. Schofield, "Estimating Software Project Effort Using Analogies," *IEEE Trans. Software Eng.,* vol. 23, no. 12, http://www.utdallas.edu/ rbanker/SE_XII.pdf, Dec. 1997.

[10] T. Menzies, D. Port, Z. Chen, J. Hihn, and S. Stukes, "Validation Methods for Calibrating Software Effort Models," *Proc. Int'l Conf. Software Eng. (ICSE),* http://menzies.us/pdf/04coconut.pdf, 2005.

[11] Z. Chen, T. Menzies, and D. Port, "Feature Subset Selection Can Improve Software Cost Estimation," *Proc. PROMISE Workshop, Int'l Conf. Software Eng. (ICSE),* http://menzies.us/pdf/05/fsscocomo.pdf, 2005.

[12] Z. Chen, T. Menzies, D. Port, and B. Boehm, "Finding the Right Data for Software Cost Modeling," *IEEE Software,* Nov. 2005.

[13] "Certified Parametric Practioner Tutorial," *Proc. 2006 Int'l Conf. Int'l Soc. Parametric Analysts (ISPA),* 2006.

[14] A. Miller, *Subset Selection in Regression,* second ed. Chapman & Hall, 2002.

[15] C. Kirsopp and M. Shepperd, "Case and Feature Subset Selection in Case-Based Software Project Effort Prediction," *Proc. 22nd SGAI Int'l Conf. Knowledge-Based Systems and Applied Artificial Intelligence,* 2002.

[16] M. Jorgensen and K. Molokeen-Ostvoid, "Reasons for Software Effort Estimation Error: Impact of Respondent Error, Information Collection Approach, and Data Analysis Method," *IEEE Trans. Software Eng.,* vol. 30, no. 12, Dec. 2004.

[17] R. Park, "The Central Equations of the Price Software Cost Model," *Proc. Fourth COCOMO Users Group Meeting,* Nov. 1988.

[18] R. Jensen, "An Improved Macrolevel Software Development Resource Estimation Model," *Proc. Fifth Conf. Int'l Soc. Parametric Analysts (ISPA),* pp. 88-92, Apr. 1983.

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

[20] V. Basili, F. McGarry, R. Pajerski, and M. Zelkowitz, "Lessons Learned from 25 Years of Process Improvement: The Rise and Fall of the NASA Software Engineering Laboratory," *Proc. 24th Int'l Conf. Software Eng. (ICSE '02),* http://www.cs.umd.edu/projects/SoftEng/ESEG/papers/83.88.pdf, 2002.

[21] T. Jones, *Estimating Software Costs.* McGraw-Hill, 1998.

[22] J. Kliijnen, "Sensitivity Analysis and Related Analyses: A Survey of Statistical Techniques," *J. Statistical Computation and Simulation,* vol. 57, nos. 1-4, pp. 111-142, 1997.

[23] D. Ferens and D. Christensen, "Calibrating Software Cost Models to Department of Defense Database: A Review of Ten Studies," *J. Parametrics,* vol. 18, no. 1, pp. 55-74, Nov. 1998.

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

[25] J.R. Quinlan, "Learning with Continuous Classes," *Proc. Fifth Australian Joint Conf. Artificial Intelligence,* pp. 343-348, 1992.

[26] R. Kohavi and G.H. John, "Wrappers for Feature Subset Selection," *Artificial Intelligence,* vol. 97, no. 1-2, pp. 273-324, 1997.

[27] M. Hall and G. Holmes, "Benchmarking Attribute Selection Techniques for Discrete Class Data Mining," *IEEE Trans. Knowledge and Data Eng.,* vol. 15, no. 6, pp. 1437-1447, Nov.-Dec. 2003.

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

[29] I.H. Witten and E. Frank, *Data Mining,* second ed. Morgan Kaufmann, 2005.

[30] S. Stukes and D. Ferens, "Software Cost Model Calibration," *J. Parametrics,* vol. 18, no. 1, pp. 77-98, 1998.

[31] S. Stukes and H. Apgar, "Applications Oriented Software Data Collection: Software Model Calibration Report TR-9007/549-1," Management Consulting and Research, Mar. 1991.

[32] S. Chulani, B. Boehm, and B. Steece, "From Multiple Regression to Bayesian Analysis for Calibrating COCOMO II," *J. Parametrics,* vol. 15, no. 2, pp. 175-188, 1999.

[33] H. Habib-agahi, S. Malhotra, and J. Quirk, "Estimating Software Productivity and Cost for NASA Projects," *J. Parametrics,* pp. 59-71, Nov. 1998.

[34] T. Ho, J. Hull, and S. Srihari, "Decision Combination in Multiple Classifier Systems," *IEEE Trans Pattern Analysis and Machine Intelligence,* vol. 16, no. 1, pp. 66-75, 1994.

[35] F. Provost and T. Fawcett, "Robust Classification for Imprecise Environments," *Machine Learning,* vol. 42, no. 3, Mar. 2001.

[36] O.T. Yildiz and E. Alpaydin, "Ordering and Finding the Best of $k > 2$ Supervised Learning Algorithms," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 28, no. 3, pp. 392-402, Mar. 2006.

[37] L. Brieman, "Bagging Predictors," *Machine Learning,* vol. 24, no. 2, pp. 123-140, 1996.

Tim Menzies received the CS degree and the PhD degree from the University of New South Wales. He is an associate professor at the Lane Department of Computer Science at the University of West Virginia and has been working with NASA on software quality issues since 1998. 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 has also worked as an object-oriented consultant in industry and has authored more than 150 publications, served on numerous conference and workshop programs, and served as a guest editor of journal special issues.

Zhihao Chen received the bachelor's and master's of computer science degrees from the South China University of Technology. He is a senior research scientist in Motorola Labs. His research interests lie in software and systems engineering, models development, and integration in general. Particularly, he focuses on quality management, prediction modeling, and process engineering. He previously worked for Hewlett-Packard, CA and EMC Corporation.

Jairus Hihn received the PhD degree in economics from the University of Maryland. He is a principal member of the engineering staff at the Jet Propulsion Laboratory, California Institute of Technology, 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 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.

Karen Lum received two BA degrees in economics and psychology from the University of California at Berkeley, and the MBA degree in business economics and the certificate in advanced information systems from the California State University, Los Angeles. She is a senior cost analyst at the Jet Propulsion Laboratory, California Institute of Technology, involved in the collection of software metrics and the development of software cost estimating relationships. She is one of the main authors of the JPL Software Cost Estimation Handbook. Publications include the Best Conference Paper for ISPA 2002: "Validation of Spacecraft Software Cost Estimation Models for Flight and Ground Systems."