Simple Software Cost Analysis: Safe or Unsafe?

Tim Menzies*
Computer Science
Portland State University.
tim@menzies.us

Dan Port⋆ Zhihao Chen‡
⋆Uni. of Hawaii, Computer Science, Manoa;
‡Center for Software Engineering,
Uni of Southern California
dport@hawaii.edu,
zhihaoch@cse.usc.edu

Jairus Hihn
Jet Propulsion Laboratory
Pasadena, USA
jairus.m.hihn@jpl.nasa.gov

ABSTRACT

Delta estimation uses changes to old projects to estimate new projects. Delta estimation assumes that new costs can be extrapolated from old projects. In this study, we show that in certain real-world data sets, there exists attributes where this assumption does not hold. We define here an automatic method to find which attributes can be safely used for delta estimation.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Time Estimation; K.6.3 [**Software Management**]: Software Process

General Terms

Algorithms, Measurement, Economics, Experimentation, Theory,Verification

Keywords

COCOMO, cost estimation, delta estimation, stability, M5, LSR

1. INTRODUCTION

In *Safe and Simple Software Cost Analysis*, Boehm presents a software effort estimation method where new projects are costed via their *delta* to previous projects [2]. The method is simple, fast, and best of all, can take full advantage of local costing information.

A premise of that *delta estimation* method is that the cost of new projects can be safely extrapolated from old projects. This paper examines this safety premise using two data sets from the PROMISE repository [8]: the 1981 *Cocomo81* data set used to derive the COCOMO-I model [1] and 60 NASA projects from described in *CocomoNASA*. All these projects come from the 1980s and 1990s.

In this study, we processed these two sets in two separate runs, as follows. In each run, we randomly sampled $\frac{2}{3}$rds the data to learn linear models. After 30 repeats of this random sub-sampling, some attributes were found to be *stable*; i.e. had a similar influence on effort in all sub-samples. We say that delta estimation is *safe*[1] when applied to these stable attributes.

However, many more attributes were *unstable*; i.e. had a very different influence on effort in the sub-samples. We say that delta estimation is *unsafe* when applied to unstable attributes that have a widely variant effect on effort.

One possibility is that instability is just an artifact of our sub-sampling policy degrading the performance of the learned models. To check that possibility, we carefully *benchmarked* the performance of our sub-samples when tested on the $\frac{1}{3}$rd of the data not used to learn the model. These models performed as well as known high water marks from the COCOMO literature; i.e. the sub-sampling policy used here does not artificially degrade the learned models.

Our results also suggest why Boehm had not seen this instability effect before. COCOMO-I's attributes are much stabler than the NASA projects. Hence, delta estimation would be less safe for projects from the NASA data than from the *Cocomo81* data. We conclude, therefore, that before conducting delta estimation, it is important to first check for attribute instability.

The rest of this paper describes COCOMO, delta estimation, our benchmark studies, and methods for checking for attribute instability. Note that this entire approach is automatic and can quickly be applied to new domains.

1.1 Digressions

Before beginning, we offer two digressions. Firstly, our original intent was to perform this study on numerous data sets from numerous industrial sites (ideally from sites that have produced multiple versions of the same project). This proved to be an impossible goal: modern corporations are very reluctant to disclose their costing data since such data often compromises their ability to effectively compete in the

*See `http://menzies.us/pdf/05safewhen.pdf` for a draft of this paper.

[1]The term “safe” is used here since Boehm uses it in the title of his paper. For other definitions of “safe”, see [6].

market place. The best we can do here is to base our study on the available COCOMO data. In the future, we hope to repeat this study on COCOMO-II. However, as far as we know, there is no large publicly available data base of COCOMO-II projects. Hence, to ensure reproducibility, this paper uses COCOMO-I data.

Secondly, we also intended to present summaries of our results compared to other researchers. This turned out not to be possible since, with the exception of our other PROMISE paper [4], we are unaware of research reporting variances in cost models as the the examples used in training change *and* the attributes in the training set also change. As to our other PROMISE paper, that work studied how *feature subset selection* can improve cost estimation predictive accuracy by intelligently selecting fewer attributes for the training set. However, that study does not study the stability of the attributes in the learned models (but such a stability study would be the logical next step).

2. COCOMO AND DELTA ESTIMATION

The COCOMO effort estimation model [1,3] measures effort in calendar months of 152 hours (and includes development and management hours). COCOMO assumes that the effort grows more than linearly on software size; i.e. $months \propto KSLOC^n$ where KSLOC is estimated directly or computed from a function point analysis. More specifically:

$$months = a * \left(KSLOC^b\right) * \left(\prod_j EM_j\right) \quad (1)$$

where EM_j is one of a set of *effort multipliers* shown in Figure 1. In COCOMO I [1], the exponent on KSLOC was a single value ranging from 1.05 to 1.2. In COCOMO II [3], the exponent b of Equation 1 was divided into a constant, plus the sum of five *scale factors* which modeled issues such as "have we built this kind of system before?".

The numeric values of the effort multipliers are shown in Figure 2. These were learned by Boehm after a regression analysis of the projects in the COCOMO I data set, followed by extensive DELPHI sessions with estimation experts [1]. Figure 1 and Figure 2 are sorted by $\frac{EM_i^{max}}{EM_i^{min}}$ where EM_i^{max} and EM_i^{min} are the largest and smallest effort multiplier values seen for EM_i. With that sort order, the effort multipliers that most *reduce* effort are shown at the top while the lower entries most *increase* effort. Note that, in COCOMO I, *sced* has a U-shaped correlation to effort; i.e. giving programmers either *too much* or *too little* time to develop a system can be detrimental.

COCOMO can be used for delta estimation as follows. An old project with known costs is used as a *baseline*. The new project to be estimated is described in terms of its deltas to the effort multipliers. The new estimate is then the product of the baseline times the effort multiplier deltas. For example, suppose the new project is estimated to have the same size as the old project. In this case, we don't need to scale the new project's cost by a factor $\left(\frac{KSLOC(new)}{KSLOC(old)}\right)^b$. However, we may need to make adjustments for other changes to the project. For example, reducing analyst capability from very high to high changes the *acap* effort multiplier from 0.71 to 0.86 (see the first line in Figure 2). Hence, according to COCOMO-I, this new project will cost $0.86/0.71 = 1.21 =$ 21% *more* than the old project [2].

increase these to decrease effort	acap: analysts capability pcap: programmers capability aexp: application experience modp: modern programing practices tool: use of software tools vexp: virtual machine experience lexp: language experience
	sced: schedule constraint
decrease these to decrease 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

Figure 1: COCOMO I effort multipliers.

	very low	low	nominal	high	very high	extra high	productivity range
acap	1.46	1.19	1.00	0.86	0.71		2.06
pcap	1.42.	1.17	1.00	0.86	0.70		2.03
aexp	1.29	1.13	1.00	0.91	0.82		1.57
modp	1.24.	1.10	1.00	0.91	0.82		1.51
tool	1.24	1.10	1.00	0.91	0.83		1.49
vexp	1.21	1.10	1.00	0.90			1.34
lexp	1.14	1.07	1.00	0.95			1.20
sced	1.23	1.08	1.00	1.04	1.10		
data		0.94	1.00	1.08	1.16		-1.23
turn		0.87	1.00	1.07	1.15		-1.32
virt		0.87	1.00	1.15	1.30		-1.49
stor			1.00	1.06	1.21	1.56	-1.56
time			1.00	1.11	1.30	1.66	-1.66
rely	0.75	0.88	1.00	1.15	1.40		-1.87
cplx	0.70	0.85	1.00	1.15	1.30	1.65	-2.36

Figure 2: COCOMO I effort multiplier values.

Note that this delta estimation approach assumes that the *only* factors that change between projects are the factors modeled in the COCOMO parameters. While the COCOMO parameters capture a wide range of issue, they are hardly complete. This is an issue well understood from the COCOMO team who continually refine and extend the COCOMO model (for example, the COCOMO II [3] model contains several more parameters than those shown in Figure 1). In the limit, this is an issue that can never be resolved: every model contains only a finite number of attributes and so may not contain an important and relevant factor. All we can offer here is this analysis is a general method of finding stable parameters within a model.

3. EVALUATION CRITERIA

3.1 Stability Criteria

Data miners can convert COCOMO-I data into *linear models* of the form:

$$X = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots \quad (2)$$

The M5 and LSR linear model learners used in this study (from the WEKA package [9]) performs M5-style parameter *pruning* [7]; i.e. they step through all the attributes removing the one with the smallest standardized coefficient until no improvement is observed in the estimate of the model error given by the Akaike information criterion. Hence, some attributes may be absent from the learned model.

Under the condition of N repeat sub-samples, we say that X_i is *unstable* if (a) it is pruned away in the majority of sub-samples (i.e. $> 0.5 * N$) or (b) there is a *large variance* (defined below) in its associated β_i value. For example, suppose we learn these three models from $N = 3$ sub-samples of some data:

$$\begin{array}{rlllll} & \underline{\beta_0} + & \underline{\beta_1 X_1} + & \underline{\beta_2 X_2} + & \underline{\beta_3 X_3} + & \underline{\beta_4 X_4} \\ \\ \textit{sub sample } 1: & 23 + & 101X_1 + & 21X_2 + & 31X_3 + & 41X_4 \\ \textit{sub sample } 2: & 25 + & 11X_1 + & & +\,30X_3 + & 42X_4 \\ \textit{sub sample } 3: & 24 + & 1X_1 + & & +\,32X_3 & \end{array} \tag{3}$$

Here, X_1 is unstable since β_1 has such a wide range. Further, X_2 is also unstable because it is usually pruned away. This example was based on $N = 3$ repeats and this is insufficient to make reliable conclusions. Appealing to the central limit theorem, the results presented below are based on $N = 30$ repeats.

In the sequel, we will use the following definition of "large variance". Given the success of the COCOMO-I model, we say that a "large variance" is one that is larger than the variances seen in COCOMO-I. This definition needs to be refined and we are working on a better definition. However, even with this approximate definition, we can show below the main result of this paper: that some attributes are very unstable indeed are hence not suitable for delta estimation.

An alternative approach to the above was proposed by one of the reviewers of this paper. Rather that check for the presence of an attribute, or the variance of its coefficient, in the learned theory another method might be to compare the variance of each independent variable with the class variable over the whole data sets. While much simpler that the study reported here, this alternate method would not study the effect of *conjunctions of attributes* on the target variable. Since our approach assesses the stability of attributes *in the context of a conjunctive model*, we have some confidence that our study catches "pair" effects where some set of attributes might be unstable in isolation but stable in conjunction.

3.2 Performance Criteria

A good software effort estimator generates predictions "close to" actual known efforts. One such measure of "closeness" is PRED(N) which is computed from the *relative error*, or RE, which is the relative size of the difference between the actual and estimated value; i.e. $RE_i = \frac{estimate_i - actual_i}{actual_i}$. Given a test set of size X, then the mean magnitude of the relative error, or MMRE, is the average percentage of the absolute values of the relative errors; i.e. $MRE_i = abs(RE_i)$ and:

$$MMRE_i = \frac{100}{X} \sum_i^X MRE_i$$

PRED(N) reports the average percentage of estimates that were within N% of the actual values:

$$PRED(N) = \frac{100}{X} \sum_i^X \left\{ \begin{array}{l} 1\ if\ MRE_i \le \frac{N}{100} \\ 0\ otherwise \end{array} \right.$$

For example, PRED(30)=50% means that half the estimates are within 30% of the actual.

This paper will compare its results with the PRED(30) values reported in the COCOMO-I and COCOMO-II studies. Baseline values for PRED(N) from COCOMO are PRED(30)= 52% (COCOMO-I [2, 163]) and PRED(30)= 69% (COCOMO-II [5]). PRED(30) was selected instead of, say, PRED(30) since the most comprehensive experiment reported in the COCOMO community comes from Chulani *et.al.* [5]) who reported the results of their 15-way *hold out* experiment in terms of PRED(30). In a such a hold-out, some fraction of the available data is selected to be a test set while the remaining data is used for testing.

Having decided to use PRED(30) from hold-out experiments, we have a problem reporting early COCOMO-I results. COCOMO-II has been studied far more rigorously than COCOMO-I and we are unaware of results from hold-out studies on COCOMO-I. The best we can say is that the COCOMO-I results are perhaps slightly inflated since they were not generated using hold-out studies (i.e. where the training set was kept separate to the test set).

4. DATA AND LEARNERS

This study applied LSR and M5 (two data miners from the WEKA toolkit [9]) to the 63 project instances in *Cocomo81* and the 60 project instances in *Cocomo-NASA*. Both data sets use the COCOMO-I attributes. LSR builds *one* linear model through the training data while M5 can build *one or more* models. Internally, M5 builds a decision tree whose leaves are linear models which apply to different zones of the parameter space. Hence, M5 outperforms LSR when the data can't be modeled as a single linear model.

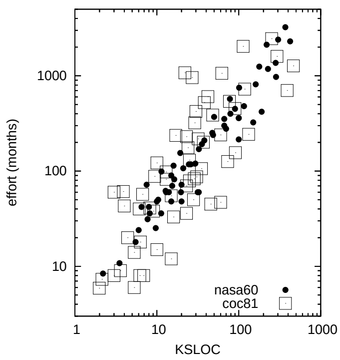

Figure 3: A log transform: LOC and effort

Nevertheless, LSR can still model non-linear data if that data is first *linearized*. For example, a *log transform* replaces all numerics with their logarithm. Data from a exponential distribution forms a straight line in a log transformed space and hence could be modeled by LSR.

For example, Equation 1 hypotheses that effort is exponential on program size. Figure 3 supports that hypothesis and shows a log transform on effort and lines of code in our two COCOMO-I data sets. As predicted by Equation 1, the relationship between these two variables is linear in the log transformed space.

Before we can explore stability and sub-sampling, we first must *certify* that our sampling and learning methods do not artificially degrade performance when learning cost estimation models from COCOMO data. Therefore, prior to the sub-sampling study, we *certified* our learners by trying various different learning strategies. Hence, we *treated* our data in several ways. Firstly, both were converted to the same set of symbols to generate *coc81* and *nasa60*. These datasets looked like this:

```
%rely    , data, ... , sced, loc,  effort
nominal  , high, ... , low,   70,     278
very_high, high, ..., low,   227,    1181
```

learner.treatment	coc81 mean	coc81 sd	nasa60 mean	nasa60 sd
lsr._num_ln	44.3	10.8	69.7	11.1
lsr._em_ln	40.0	9.7	68.5	12.5
m5._num_ln	39.7	13.7	73.5	10.7
m5._em_ln	38.4	9.2	69.7	10.5
m5._em_loc_ln	21.7	8.5	60.5	9.6
lsr._em_loc_ln	21.7	8.5	60.5	9.6
m5._num_loc_ln	20.6	6.9	55.3	11.7
lsr._num_loc_ln	20.6	6.9	55.3	11.7
m5._em	15.4	8.4	40.8	14.4
m5._num	13.7	8.7	41.0	11.6
m5._num_loc	11.7	6.9	41.5	8.9
m5._em_loc	11.7	6.9	42.0	12.7
lsr._num_loc	11.3	6.7	41.2	8.4
lsr._em_loc	11.3	6.7	40.2	14.1
lsr._num	9.4	6.7	31.0	10.8
lsr._em	7.9	5.8	28.7	13.0

Figure 4: Mean and standard deviations of PRED(30) seen in 30 repeats of using *coc81* and *nasa60*.

...

Next, for $X \in \{coc81, nasa60\}$, the following treatments were applied:

X_num: The symbols *very low, low, nominal, etc* were replaced with 0.8,0.9,1,etc. Under these substitutions, *nominal*s effect the COCOMO-I output of Equation 1 by a factor of one.

X_em: The attribute values were replaced by their associated effort multiplers from Figure 2.

X_num_loc, X_em_loc: All the effort multiplier attributes were removed, leaving just *lines of code* and *effort*.

X_num_ln, X_em_ln, X_num_loc_ln, X_em_loc_ln: A log transform was applied to the above data sets.

To certify the competency of our method, we benchmarked our results with prior landmark COCOMO results [5]. Using the same methodology as those prior results, we ran M5 and LSR over the treated data sets using 30 repeats of a $\frac{2}{3}rds/\frac{1}{3}rd$ hold-out experiment. The results were converted to *win/loss* tables as follows. For each of the 30 selections, the training and test sets were transformed them into *X_num, X_em,* etc; then run through M5 and LSR. This generated a table of 30 results, with separate columns $1 \ldots C$ for each combination of {M5,LSR}×*X_num, X_em,* etc. For each pair of columns $\{i, j\} \in \{1 \ldots C\}, j > i$, a two-tailed t-test was performed to check if the mean of one column was significantly different to the other. If not, then a "tie" was declared. Otherwise, the means were compared numerically to declare a "win" or a "loss". The resulting "win"/"loss"/"tie" counts were then sorted by "win-losses".

The "win-loss" tables let us select the combination of learner (M5 or LSR) and the treatment (*X_num, X_em,* etc) that performs the best. This best combination was then used in the sub-sampling study where, 30 times, we generated $\frac{2}{3}$rd/$\frac{1}{3}$rd train/test sets; then collected the PRED(30) scores seen when the models learned from the training set were applied to the test set.

5. RESULTS

This certification phase of this work required 960 runs:

$$2\ learners * 2\ data\ sets\ * 8\ treatments * 30\ repeats$$

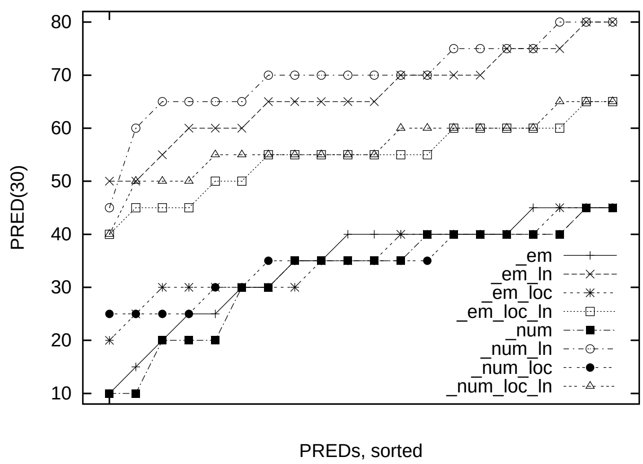

Figure 5: Results from 20 hold-outs of M5 on *nasa60*.

Each run generated an effect estimator. For example, here is one effect estimator learned from *nasa60_em_ln* by LSR:

$$\begin{aligned} act_effort = \; & -0.8474 * rely + 2.7259 * data + 2.9451 * time + \\ & -0.6964 * stor + 0.882 * turn + 2.8169 * acap + \\ & -6.6709 * vexp + 3.5478 * lexp + -1.8933 * modp + \\ & 1.1274 * loc + 1.4098 \end{aligned}$$

Figure 4 shows a summary of the certification results. The mean PREDs for *coc81* were always lower than the mean PREDs from *nasa60* (on average, by about 20%). Figure 5 shows that the variance in the estimates can be quite large: in 30 repeated trials on M5 on *nasa60*, the max and min PRED(30)s seen in out treated data sets can be as large as 45%. Hence, in order to compare our means, we turn to the t-test results shown in Figure 6 and Figure 7. In terms of "wins-losses", the pattern of results is the same for both *coc81* and *nasa60*:

- Linearization *always* beat *non-linearization*
- PREDs found using *just* lines of code were *always* worse that using all the effort estimators.
- The *best performance* was seen using linearized data that included the effort multiplier attributes.

In the top-performing cases, in terms of "wins-losses":

- Using simple numbers like 0.8,0.9,1, etc did as well as using the effort multipliers of Figure 2.
- Neither M5 or LSR was a clear winner.

From the win-loss tables, we see that the best combination of learner and treatments is either (M5 or LSR) and (_num_ln or _em_ln). LSR produces a simpler output than M5 so the following results just use LSR. Figure 8 and Figure 9 show a Equation 3-style analysis of the β_i values seen in the 30 hold-outs with LSR on _num_ln and _em_ln. Note that these figures only show effort multipliers that were found in the *majority* of the 30 hold-outs; i.e. if an effort multipler variable appears *less than* 16 times, it is not shown.

The important features of Figure 8 and Figure 9 are:

- LOC is always present, with low β_i deviation.

learner.treatment	Wins - Loses	Win	Loss	Tie
m5._num_ln	12	12	0	3
m5._em_ln	12	12	0	3
lsr._num_ln	12	12	0	3
lsr._em_ln	12	12	0	3
m5._num_loc_ln	3	7	4	4
m5._em_loc_ln	3	7	4	4
lsr._num_loc_ln	3	7	4	4
lsr._em_loc_ln	3	7	4	4
m5._em	0	4	4	7
m5._num_loc	-7	1	8	6
m5._num	-7	1	8	6
m5._em_loc	-7	1	8	6
lsr._num_loc	-8	1	9	5
lsr._em_loc	-8	1	9	5
lsr._num	-9	0	9	6
lsr._em	-14	0	14	1

Figure 6: T-tests on *coc81*: 99% level

learner.treatment	Wins - Loses	Win	Loss	Tie
m5._num_ln	12	12	0	3
m5._em_ln	12	12	0	3
lsr._num_ln	12	12	0	3
lsr._em_ln	10	10	0	5
m5._em_loc_ln	5	8	3	4
lsr._em_loc_ln	5	8	3	4
m5._num_loc_ln	4	8	4	3
lsr._num_loc_ln	4	8	4	3
m5._num_loc	-6	2	8	5
m5._num	-6	2	8	5
m5._em_loc	-6	2	8	5
lsr._num_loc	-6	2	8	5
m5._em	-6	2	8	5
lsr._em_loc	-7	1	8	6
lsr._num	-13	0	13	2
lsr._em	-14	0	14	1

Figure 7: T-tests on *nasa60*: 99% level

coc81	n	mean	sd
loc	30	1.2	0.1
cplx	24	1.4	0.3
time	16	1.9	0.4
pcap	29	1.7	0.5
acap	29	2.1	0.5
rely	30	2.0	0.6
sced	25	2.9	0.7
virt	22	2.3	0.8
vexp	24	3.0	1.2

nasa60	n	mean	sd
loc	30	1.1	0.0
stor	22	-0.8	0.8
time	29	2.5	0.8
cplx	16	1.7	1.0
acap	28	2.7	1.0
data	26	3.1	1.2
turn	17	1.8	1.5
modp	16	-1.8	1.6
vexp	26	-5.6	2.3
lexp	16	2.5	3.0

Figure 8: LSR β_i values from *_em_ln data.

coc81	n	mean	sd
loc	30	1.2	0.1
cplx	24	2.0	0.5
sced	29	-3.0	0.6
stor	22	2.5	0.6
pcap	28	-3.2	0.7
rely	30	3.3	0.8
vexp	30	-3.4	0.8
acap	30	-3.8	1.0

nasa60	n	mean	sd
loc	30	1.1	0.0
data	22	1.8	1.0
time	28	4.4	1.2
modp	25	2.6	1.4
stor	24	-1.2	1.4
acap	27	-5.2	1.7
sced	25	2.5	2.6
vexp	22	4.6	2.8
turn	16	1.6	2.9

Figure 9: LSR β_i from *_num_ln data.

- Nearly half the attributes defined in COCOMO-I, are missing (i.e. are unstable) in both data sets
- The list of missing attributes varies from experiment to experiment; e.g. pcap appears in the majority of the *coc81* results but not at all in results from the *nasa60* data sets.

Also, the *coc81* attributes are stabler than the *nasa60* attributes:

- Most of the *coc81* β_i standard deviations are less than 1.0 while most of the *nasa60* β_i standard deviations are greater than 1.0 (and, in five cases, greater than 2.0).
- The stable *coc81* attributes are *well-behaved* in the sense that their β_i standard deviation *only increases* when the mean β_i values increases.
- The stable *nasa60* attributes are not so *well-behaved*. Large β_i standard deviations can be found even when the β_i mean is quite small; e.g. turn has one β_i mean of 1.6 but a β_i standard deviations of 2.9(!).

6. DISCUSSION

We set out with the intent of identifying stable attributes that can be safely used in delta estimation. Along the way, we needed to define some baselines for COCOMO-style effort estimation. Consequently, our results comment both on COCOMO effort estimation and delta estimation.

Adequacy of our tools: If we could not have reproduced known baselines from the COCOMO research, then our results would have been questionable. The tools described above used COCOMO-I data and LSR to achieve comparable results to known benchmarks for COCOMO-I of PRED(30)=52% (see Figure 4)[2].

Merits of just using LOC: Figure 3 suggests that lines of code might be enough information to perform adequate software cost estimation. Figure 6 and Figure 7 shows that this is not necessarily the case. In terms of "wins-loses", estimates based on just lines of code always performed worse than linearized data sets containing COCOMO-I effort multipler attributes.

Merits of linearization: Equation 1 predicts that effort is exponential on program size. Our results supported this core premise of the COCOMO research. In Figure 6 and Figure 7, linearized data sets always performed non-linearized data sets. Hence, this study endorses the core COCOMO assumption that a *single* continuous exponential function is adequate for modeling software effort.

Merits of multi-segmented models: This study did not find any value is using a combination of *multiple* continuous exponential functions to model software effort. Recall that in Figure 6 and Figure 7, M5 never out-performed LSR on the linearized data. That is, a multi-segmented model (in log transformed space) did not do better than a model with a single-segment.

Merits of stratification: $Nasa60$ comes just from the aerospace domain while *coc*81 comes from many domains including finance, engineering, etc. The stratification hypothesis [3] claims that specialized subset (like *nasa*60) should generate higher PREDs than general data sets (like *coc*81). Figure 4 lends support to the stratification hypothesis since the PRED(30) means from the stratified *nasa60* data sets are all 20% (approx) higher than the PRED(30) means. However, curiously, this higher PRED comes with a cost: more instability in the β_i values. In the future we plan to

[2]To be precise, we used hold-out studies and the COCOMO-I benchmark study did not. Also, our exact best results were a mean PRED(30) of 44.3% with a standard deviation of 10.8; i.e. the COCOMO-I results were within 70% of one standard deviation to our results.

explore this strange disconnect where instability does *not* mean lower PREDs.

Merits of effort multiplier attributes: In Figure 6 and Figure 7, the data sets using all the effort multiplier attributes defined in Figure 1 always improved PRED over just using LOC. Hence, using *some* effort multiplier attributes is useful. However, using *all* of them may not be. Figure 8 and Figure 9 showed that it is possible to ignore some of the effort multiplier attributes are still reach the adequate performance levels seen in Figure 4. Strangely, the ignorable attributes are not fixed (recall in Figure 8 and Figure 9 how different stable attributes appeared in the different tables). Hence, it is still necessary to collect *all* the COCOMO attributes even if they are not all used later on.

Applying COCOMO in novel domains: In Figure 6 and Figure 7, the data sets using the Figure 2 values did not perform better than data sets just using simpler values such as very low=0.8, low=0.9, nominal=1, etc. This means that an adequate first-pass approximation for development effort can be computed *without* requiring Figure 2-like values calibrated from numerous past projects. This is an important result since it means that COCOMO can be applied to totally novel domains (e.g. cost models for autonomous deep-space robots) without needing a historical record of past, similar projects (which, for novel domains, may be non-existent).

Detecting stable attributes: The methods used above can automatically detect stable attributes, That check means generating Figure 4 to Figure 9. Given our current tools, this check takes around 30 minutes, on a standard LINUX machine. If we adopt the *coc81* results as a baseline for stability, then could declare an attribute *unstable* if the associated β_i standard deviation is greater than 1.2; i.e. the maximum COCOCMO-I β_i standard deviation seen anywhere in Figure 8 or Figure 9.

When delta estimation is safe: Delta estimation is safe when the extrapolation from old to new projects is based on changes to stable attributes. In the case of Figure 8, there six stable attributes with a β_i standard deviation within 1.2. Hence, for that experiment, the attributes that can be safely used for delta estimation is {loc, stor, time, cplx,acap, and data}.

This list of stable attributes may not repeat in other domains (recall how the set of ignorable attributes changes in all the tables of Figure 8 and Figure 9). Hence, if the reader wishes to perform delta estimation in their own domain, it is highly recommended that they first find the attributes that are stable in their own domain.

Acknowledgments

This research was conducted at Portland State University, the Jet Propulsion Laboratory, and the University of Hawaii sponsored, in part, by the NASA Office of Safety and Mission Assurance under the Software Assurance Research Program led by the NASA IV&V Facility. 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.

7. REFERENCES

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

[2] B. Boehm. Safe and simple software cost analysis. *IEEE Software*, pages 14–17, September/October 2000. Available from `http://www.computer.org/certification/beta/Boehm_Safe.pdf`.

[3] Barry Boehm, Ellis Horowitz, Ray Madachy, Donald Reifer, Bradford K. Clark, Bert Steece, A. Winsor Brown, Sunita Chulani, and Chris Abts. *Software Cost Estimation with Cocomo II.* Prentice Hall, 2000.

[4] Zhihao Chen, Tim Menzies, and Dan Port. Feature subset selection can improves software cost estimation. In *Proceedings, PROMISE workshop, ICSE 2005*, 2005.

[5] S. Chulani, B. Boehm, and B. Steece. Bayesian analysis of empirical software engineering cost models. *IEEE Transaction on Software Engineerining*, 25(4), July/August 1999.

[6] N. Leveson. *Safeware System Safety And Computers.* Addison-Wesley, 1995.

[7] J. R. Quinlan. Learning with Continuous Classes. In *5th Australian Joint Conference on Artificial Intelligence*, pages 343–348, 1992. Available from `http://citeseer.nj.nec.com/quinlan92learning.html`.

[8] J. Sayyad Shirabad and T.J. Menzies. The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering, University of Ottawa, Canada, 2005. Available from `http://promise.site.uottawa.ca/SERepository`.

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