Validation Methods for Calibrating Software Effort Models

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
Sherry Stukes
Jet Propulsion Laboratory, Pasadena
jairus.m.hihn@jpl.nasa.gov, sherry.stukes@jpl.nasa.gov

ABSTRACT

COCONUT calibrates effort estimation models using an exhaustive search over the space of calibration parameters in a COCOMO I model. This technique is much simpler than other effort estimation method yet yields PRED levels comparable to those other methods. Also, it does so with less project data and fewer attributes (no scale factors). However, a comparison between COCONUT and other methods is complicated by differences in the experimental methods used for effort estimation. A review of those experimental methods concludes that software effort estimation models should be calibrated to local data using incremental hold-out (not jack knife) studies, combined with randomization and hypothesis testing, repeated a statistically significant number of times.

Categories and Subject Descriptors

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

General Terms

Management, measurement, economics, experimentation

Keywords

COCOMO, calibration, incremental cross-validation

1. INTRODUCTION

Software effort models work better when calibrated with local data [2, 5, 8, 9, 12, 14, 17, 18]. However data collection from industry is notoriously slow and most industrial sites lack the resources to conduct extensive calibration studies. For example, few industrial sites that could repeat a COCOMO II-style calibration experiment [2]:

- An initial regression analysis was conducted on 83 projects to generated the COCOMO I model [1];
- Further data collection found 78 more projects;
- A DELPHI panel convened where experts offered their best judgment on factors controlling software costs;
- A Bayesian calibration technique integrated the DELPHI results with data from the 83+78 projects.

To shortcut the development time of an effort estimation model, the COCOMO team added *calibration parameters* a and b in their model. They recommended "having at least... 10 points for calibration both the multiplicative constant a and the baseline exponent b". [1, p175]. Shepperd and Schofield make a similar conclusion, saying that their effort estimation models stabilized after seeing 15 projects [14]. However, the evidence for these claims is not conclusive. Shepperd and Schofield only repeated their stability studies three times and the variance in their results is rather large. Also, when the COCOMO team tried two experiments with calibration from 8 projects, the first one succeeded and the second failed. They remark:

> ..the local calibration of (parameters) is more beneficial when there is data on many projects versus just 8 [2, p180].

This observation was the starting point for this paper. If eight projects is too little data then when is enough data enough? Is 10 or 15 enough? If "many projects" are required, how many is "many"?

Clearly, more experimentation is required to understand the calibration of software effort estimation models. In defining a validation experiment for our own COCONUT calibration tool, we reviewed prior experiments on effort estimation. Comparing any two of those experiments is difficult since their experimental methods can be very different. Hence, we defined a new procedure that combines the best features of past research into calibrating effort estimation models. We commend this procedure to the effort estimation community since it controls for certain effects that can complicate calibration. Further, it can report the minimum point at which enough project data is enough.

The rest of this paper reviews the need for local calibration, documents the effort models used in our work, discusses

*For a draft of this paper, see `http://menzies.us/pdf/04coconut.pdf`

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICSE'05, May 15–21, 2005, St. Louis, Missouri, USA.
Copyright 2005 ACM 1-58113-963-2/05/0005 ...$5.00.

rely	data	cplx	time	stor	virt	turn	acap	aexp	pcap	vexp	lexp	modp	tool	sced	ksloc	actual effort (months)
h	l	h	n	n	l	l	n	n	n	n	h	h	n	l	2.2	8.4
h	l	h	n	n	l	l	n	n	n	n	h	h	n	l	3.5	10.8
h	l	h	n	n	l	l	n	n	n	n	h	h	n	l	5.5	18
n	l	h	n	n	l	l	h	vh	h	n	h	n	n	n	6	24
h	l	h	n	n	l	l	n	n	n	n	h	h	n	l	9.7	25.2
h	l	h	n	n	l	l	n	n	n	n	h	h	n	l	7.7	31.2
n	l	h	n	n	l	l	h	vh	n	n	l	n	n	n	11.3	36
h	l	h	n	n	l	l	n	n	n	n	h	h	n	l	8.2	36
h	n	h	n	n	n	n	n	n	n	n	n	n	n	n	6.5	42
n	n	h	n	n	n	n	n	n	n	n	n	n	n	n	8	42
n	l	h	n	n	l	l	h	vh	h	n	h	n	n	n	20	48
n	n	n	n	n	n	n	n	h	h	n	n	n	n	n	10	48
n	l	h	n	n	l	l	h	vh	h	n	h	n	n	n	15	48
h	l	h	n	n	l	l	n	n	n	n	h	h	n	l	10.4	50
n	n	h	n	n	n	n	n	n	n	n	n	n	n	n	13	60
h	l	h	n	n	l	l	n	n	n	n	h	h	n	l	14	60
h	l	h	n	n	l	l	n	n	n	n	h	h	n	l	19.7	60
n	l	h	n	xh	l	l	h	h	n	n	h	n	n	n	32.5	60
n	l	h	n	n	l	l	h	h	h	n	h	n	n	n	31.5	60
n	vh	h	vh	vh	l	h	vh	h	n	l	h	vh	vh	l	12.8	62
n	vh	h	vh	vh	l	h	vh	h	n	l	h	vh	vh	l	15.4	70
n	l	h	n	n	l	l	h	vh	vh	n	h	n	n	n	20	72
h	l	h	xh	xh	l	h	h	h	h	n	h	h	h	n	7.5	72
n	vh	h	vh	vh	l	h	vh	h	n	l	h	vh	vh	l	16.3	82
h	n	n	h	n	n	n	n	h	h	n	n	n	n	n	15	90
n	h	h	vh	n	n	h	h	h	h	n	h	l	l	h	11.4	98.8
vh	n	xh	h	h	l	l	n	h	n	n	n	l	h	n	21	107
n	n	h	h	n	n	n	n	n	n	n	n	n	n	n	16	114
h	l	h	n	n	l	l	n	n	n	n	h	h	n	l	25.9	117.6
h	l	h	n	n	l	l	n	n	n	n	h	h	n	l	24.6	117.6
h	l	h	n	n	l	l	n	n	n	n	h	h	n	l	29.5	120
n	h	h	vh	n	n	h	h	h	h	n	h	l	l	h	19.3	155
n	vh	h	vh	vh	l	h	vh	h	n	l	h	vh	vh	l	32.6	170
n	vh	h	vh	vh	l	h	vh	h	n	l	h	vh	vh	l	35.5	192
h	n	h	n	n	n	n	n	h	h	n	n	n	n	n	38	210
n	l	h	n	n	h	l	h	h	h	l	vl	n	n	n	100	215
n	vh	h	vh	vh	l	h	vh	h	n	l	h	vh	vh	l	48.5	239
n	n	h	n	h	n	n	h	h	n	n	h	h	n	h	47.5	252
n	h	vh	n	n	l	n	h	n	vh	l	n	h	n	l	70	278
h	l	h	n	n	l	l	n	n	n	n	h	h	n	l	66.6	300
n	l	h	n	xh	l	l	h	vh	vh	n	h	n	n	n	150	324
h	l	h	n	n	l	l	n	n	n	n	h	h	n	l	66.6	352.8
n	l	h	n	n	l	l	h	vh	vh	n	h	n	n	n	100	360
n	l	h	n	n	l	l	h	n	n	n	vl	n	n	n	100	360
h	n	h	h	h	l	h	n	h	n	n	n	l	vh	n	50	370
h	h	n	n	n	l	l	n	h	h	n	h	n	n	n	79	400
n	n	n	n	n	l	n	h	vh	vh	l	h	h	n	n	190	420
n	n	h	n	n	n	n	n	n	h	n	h	h	h	n	90	450
h	l	h	n	n	l	l	n	n	n	n	h	h	n	l	115.8	480
n	h	h	vh	n	n	h	h	h	h	n	h	l	l	h	78	571.4
h	n	vh	h	h	l	h	h	n	n	h	h	l	vh	h	101	750
n	vh	h	vh	vh	l	h	vh	h	n	l	h	vh	vh	l	161.1	815
h	h	l	n	n	n	h	h	h	n	n	n	h	n	n	284.7	973
vh	h	h	vh	vh	n	n	vh	vh	vh	n	h	h	h	l	227	1181
n	h	h	vh	h	l	h	h	n	h	l	h	h	n	l	177.9	1248
n	h	l	n	n	h	n	h	h	n	n	n	h	h	n	282.1	1368
h	n	h	h	h	l	h	n	h	n	n	n	l	vh	n	219	2120
l	n	n	n	n	l	l	h	h	vh	n	h	l	l	h	423	2300
h	l	h	n	n	l	l	n	n	h	n	n	h	vl	n	302	2400
h	n	h	h	h	l	h	n	h	n	n	n	l	vh	n	370	3240

KEY:
xh= extra high
vh= very high
h= high
n= nominal
l= low
vl= very low

Figure 1: NASA effort data used in this study. Available on-line at the *PROMISE* repository of public domain software engineering data sets: `http://promise.site.uottawa.ca/SERepository/datasets/cocomonasa.arff`.

prior experiments in effort estimation, and presents results from our preferred method. Based on those results, we speculate that the COCONUT tool is an effective and simpler alternative to traditional effort estimation tools. This paper is offered as a baseline result with the hope that other researchers will try to reproduce and out-perform our results.

Before beginning, we digress for a comment on the problems associated with collecting industrial software engineering data. In the future, we hope to apply our methods to data from many sources. However, for now, the methods of this paper have only been applied to the historical NASA data of Figure 1. There is a good reason for this. Collecting such new data is very difficulty. Software projects are notoriously difficult to control[1]. Corporations are therefore reluctant to expose their own software development record to public scrutiny.

Given this data shortage, it is important that the available data is put to best use. Therefore, much of this paper is devoted to a methodological discussion on *how* to best calibrate a cost model.

2. THE CASE FOR LOCAL CALIBRATIONS

Before discussing *how* to perform local calibration, this section discusses *why* such calibrations are useful.

The case *against* calibration is statistical: if a large database of examples exists, then the distributions in that large database should include the particulars of some current project. Hence, it would seem advantageous to *not* calibrate in order to make full use of the knowledge in the large database of examples.

The problem with this argument is that it assumes a database large enough to capture the underlying distributions of the software development process. Figure 2 suggests that we do not have such a "large enough" database.

That figure shows the percentage of times an attribute was selected by a N=4 hill climbing search. In hill climbing, if adding a attribute to a set of currently selected attributes does not improve the score (in this, effort estimation accuracy), then that set is marked "stale". If that set remains "stale" even after N more additions, then the last $N + 1$ added attributes are rejected and the hill climber searches elsewhere. Otherwise, the attributes in the non-stale set are selected as being influential on the score.

This hill-climbing search was conducted on COCOMO II data divided into pre- and post-1990 projects. Ten experiments were performed with each of the two divisions. Each experiment performed hill climbing on a randomly selected 90% of the division data. Figure 2 only shows the *most often influential attributes*; i.e. those that were selected in five or more of the experiments in *either* the pre- or post-1990 sample.

Some of those attributes were selected with similar frequencies in both samples (*pcon*, *site*, *size*). However, the remaining influential attributes *had very different influence on effort* in the pre- and post-1990 samples. Some of the changes we can't explain such as the dramatic drop in the importance of *pcap* (programmer capability). Other chances correspond to general trends in the computer industry. For example, the *team* cost driver (i.e. team cohesion) has become more significant. This is not unexpected, given the boom in knowledge management, experience management and organization learning since 1990. Another interesting change is that the *stor* cost driver (i.e. main memory storage constraint) has become dramatically less important. After all the development in hardware and software for the last two decades, main storage is more readily available.

[1]Recall the 1995 report of the Standish group that described a $250 billion dollar American software industry where 31% of projects were canceled and 53% of projects incurred costs exceeding 189% of the original estimate [16].

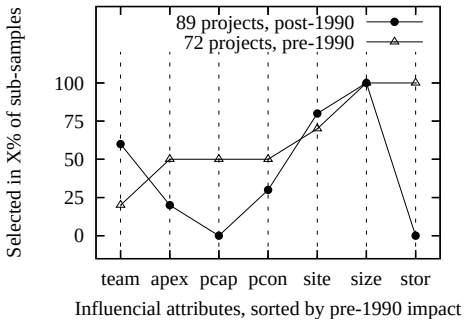

Figure 2: Influence of various COCOMO II attributes within two sub-samples of the 2004 database found by a hill climbing (N=4) search.

Regardless of the reason for the change in the influence of the attributes, the general lesson of Figure 2 is that data collected in one context (e.g. pre-1990) may not be completely relevant to some other context. An often repeated result in effort estimation is that *stratification* of the available data into similar projects, can improve the effectiveness of estimation. Similar projects have less variation and so can be easier to calibrate. A commonly used measure of predictive accuracy is the "PRED" value defined is §4. Chulani et.al. [4] report that stratifying data can improved their "PRED" values anywhere between 5% to 12%. Shepperd and Schofield report even more dramatic improvements in "PRED" from 4% to 44% [14].

3. EFFORT ESTIMATION MODELS

Having made the case that local calibration is useful, this section describes the BASE and COCOMO I effort estimation model used in our calibration experiments.

There are many effort estimation models[2] and the one we used here was chosen based on the available publishable data. Our intent was to define a repeatable effort estimation experiment so that others may repeat to refute or improve our results. The COCOMO II data is *not* publishable since it was collected on condition of confidentiality with the companies supplying the data.

Figure 1 showed the data used in this study. That data comes from 60 NASA projects from different centers for projects from the 1980s and 1990s. Since it is from NASA, it is stratified to just aerospace applications. The data is in COCOMO I format so it lacks the scale factors (defined below) used in COCOMO II.

COCOMO 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. In our experiments, we will model this assumption two ways. The BASE model assumes:

[2]See the excellent review in [6, Chapter 2].

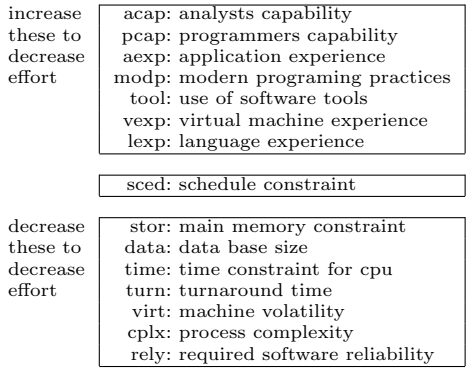

Figure 3: COCOMO I effort multipliers.

	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		1.67
aexp	1.29	1.13	1.00	0.91	0.82		1.57
modp	1.24.	1.10	1.00	0.91	0.82		1.34
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		
stor			1.00	1.06	1.21	1.56	-1.21
data		0.94	1.00	1.08	1.16		-1.23
time			1.00	1.11	1.30	1.66	-1.30
turn		0.87	1.00	1.07	1.15		-1.32
virt		0.87	1.00	1.15	1.30		-1.49
cplx	0.70	0.85	1.00	1.15	1.30	1.65	-1.86
rely	0.75	0.88	1.00	1.15	1.40		-1.87

Figure 4: COCOMO I effort multiplier values.

$$months = a * KSLOC^b. \quad (1)$$

where a and b are domain-specific parameters and KSLOC is estimated directly or computed from function points.

In our experiments, this BASE equation is compared to a more sophisticated model:

$$months = a * \left(KSLOC^b\right) * \left(\prod_j EM_j\right) \quad (2)$$

Equation 2 is Boehm's COCOMO I model [1]. EM_j is one of a set of *effort multipliers* shown in Figure 3.

In COCOMO I, the exponent on KSLOC was a single value ranging from 1.05 to 1.2. In COCOMO II, the exponent b of Equation 2 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 COCOMO II effort multipliers are similar but COCOMO II dropped one of the Figure 3 parameters; renamed some others; and added a few more (for "required level of reuse", "multiple-site development", and "schedule pressure").

The numeric values of the effort multipliers are shown in Figure 4. These were learnt by Boehm after a regression analysis of the projects in the COCOMO I data set [1]. The multipliers fall into three groups: those that are *positively* correlated to more effort; those that are *negatively* correlated to more effort; and a third group containing just schedule information. 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.

The last column of Figure 4 is $\frac{EM_{max}}{EM_{min}}$ and shows the overall effect of a single effort multiplier. For example, moving *acap* (analyst experience) from very low to very high will *most decrease* effort while moving *rely* (required reliability) from very low to very high will *most increase* effort.

There is more to COCOMO that the above description. The COCOMO II text [2] is over 500 pages long and offers all the details needed to implement data capture and analysis of COCOMO in an industrial context.

4. PERFORMANCE MEASURES

Our preferred experiment is a combination of the best techniques seen in the effort estimation calibration experiments of Figure 5. Those experiments all take the form:

$$\{Repeats, Train, Test\}$$

That is, *Repeats* times the available data is divided into a $Train$ set and a $Test$ set. Some tool is applied to the training set to learn a model or calibrate parameters within a model. The model or parameters are then frozen and applied to the $Test$ set.

When the $Train$ and $Test$ set do not intersect this is called a *holdout* study. Holdout studies check the utility of the calibrated parameters using data not used during calibration. If the calibrations are tested on the training set, this is not a holdout study. Two non-holdout studies are shown in Figure 5: see the $|Test| = 0$ entries. An extreme form of holdout studies are the *jack knife* studies used in rows $\{d \ldots l\}$ of Figure 5) where *every* example is removed one at a time, training occurs on the remaining data, then testing is conducted on the single holdout example.

The results of Figure 5 are expressed in terms of MMRE and PRED(N) (for notes on other performance measures, see [19, chapter 5]). MMRE and PRED are computed from the *relative error*, or RE, which is the relative size of the difference between the actual and estimated value:

$$RE_i = \frac{estimate_i - actual_i}{actual_i}$$

Given a data set of of size D, a $Train$ing set of size $(X = |Train|) \leq D$, and a *test* set of size $T = D - |Train|$, then the mean magnitude of the relative error, or MMRE, is the percentage of the absolute values of the relative errors, averaged over the T items in the $Test$ set; i.e.

$$MRE_i = abs(RE_i)$$
$$MMRE_i = \frac{100}{T} \sum_i^T MRE_i$$

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

$$PRED(N) = \frac{100}{T} \sum_i^T \begin{cases} 1 \; if \; MRE_i \leq \frac{N}{100} \\ 0 \; otherwise \end{cases}$$

For example, e.g. PRED(30)=50% means that half the estimates are within 30% of the actual. Shepperd and Schofield comment that:

> MMRE is fairly conservative with a bias against overestimates while Pred(25) will identify those prediction systems that are generally accurate but occasionally wildly inaccurate [14, p736].

The rows labeled $\{d, i\}$ in Figure 5 illustrate how stratification can improve effort estimation. In those experiments,

id	Holdout?	Repeats	$\|Train\|$	$\|Test\|$	Best Results	Notes	Reference
a	y	3	105	10	PRED(25)=75%	Wittig&Finnie studied neural networks predicting software effort that yielded an average PRED(25) of 75%	[20]
b	n				MMRE=70%	Back propagation neural nets	[15]
c	n	1	8	0	PRED(20)=100%	In the COCOMO II experiment discussed in the introduction, a PRED(20)=75% model was calibrated to a PRED(20)=100%.	[2] p175-181
d_0	y	28	27	1	PRED(25)=21%	The Mermaid data: unstratified, estimation by analogy	[14]
d_1	y	?	?	1	PRED(25)=39%	The Mermaid E data: stratified, estimation by analogy	[14]
d_2	y	?	?	1	PRED(25)=25%	The Mermaid N data: stratified, estimation by analogy	[14]
e	y	21	20	1	PRED(25)=23%	The Real-time1 data: unstratified, estimation by analogy	[14]
f	y	24	23	1	PRED(25)=33%	The Albrecht data: unstratified, estimation by analogy	[14]
g	y	38	37	1	PRED(25)=39%	The Finnish data: unstratified, estimation by analogy	[14]
h	y	15	14	1	PRED(25)=40%	The Kemerer data: unstratified, estimation by analogy	[14]
i_0	y	77	76	1	PRED(25)=42%	The Desharnais data: unstratified, estimation by regression	[14]
i_1	y	?	?	1	PRED(25)=47%	The Desharnais 1 data: stratified, estimation by analogy	[14]
i_2	y	?	?	1	PRED(25)=48%	The Desharnais 2 data: stratified, estimation by regression	[14]
i_3	y	?	?	1	PRED(25)=70%	The Desharnais 3 data: stratified, estimation by analogy	[14]
j	y	21	20	1	PRED(25)=43%	The Atkinson data: unstratified, estimation by regression	[14]
k	y	18	17	1	PRED(25)=44%	The Telecom 1 data: unstratified, estimation by analogy	[14]
l	y	33	32	1	PRED(25)=51%	The Telecom 2 data: unstratified, estimation by analogy	[14]
m	n	1	83	0	PRED(30)=64%	Boehm improved the COCOMO-II 1997 PRED(30) results from 52% to 64%, before and after stratification.	[2]
n	y	15	121	40	PRED(30)=69%	Chulani (of the COCOMO II team) conducted experiments in calibrates the COCOMO II 2000 software effort model using a Delphi panel and a Bayesian calibration method.	[2]
o	y	3	67	10	MMRE=21%	Mair et.al. compared the performance of adaptive neural nets (ANN), case-based reasoning, rule induction, and linear regression. In that study, the ANN produced the best MMRE's of 21,53,66% in each of the three runs.	[13]
p	y	10	99	10	PRED(25)=83.3%	Boetticher reported a $10 * 99/10$ study that yielded software effort estimates with an average PRED(25)/MMRE across the ten repeats of of 83.3% /14.7% (respectively).	[3]
q	y	30	56 52 48 44 ...	4 8 12 16 ...	PRED(30)=70% PRED(30)=70% PRED(30)=70% PRED(30)=70%		This study.

Figure 5: A sample of effort estimation experiments in the literature. Missing numbers, marked with "?", come from a lack of information in the source document.

the unstratified data of $\{d_0, i_0\}$ was divided into related subsets to form $\{d_1, d_2\}$ and $\{i_1, i_2, i_3\}$. The d_0/d_1 comparison shows the least improvement: 21% to 25% while the $\{i_0/i_3\}$ comparison show the greatest improvement: 42% to 70%.

An interesting variant on the standard calibration experiment is the *incremental holdout experiment* used by (e.g.) Shepperd and Schofield which explores "the dynamic behavior of effort prediction by simulating the growth of a dataset over time" [14, p741]. That is, as the study progresses, more and more of the data is moved from test to training in units of size δ. That is, holdout experiments convert a $\{Repeats, Train, Test\}$ experiment into

$$\{Repeats, Train, Test, \delta\}$$

For example, if at every step in a holdout study, one example is moved from $Test$ to $Train$r then such a

$$\{Repeats, |Train| = 1, |Test| = 100, \delta = 1\}$$

study could look like this:

```
while( Repeats-- )
    randomizeOrder
    train on item  1     test on items 2..100
    train on items 1,2,  test on items 3..100
    train on items 1,2,3 test on items 4..100
    ..
    train on items 1..97 test on item 98,99,100
    train on items 1..98 test on item 99,100
    train on items 1..99 test on item 100
```

5. EXPERIMENTAL METHOD

Each of the Figure 5 studies has merit, but their results are hard to compare since they mostly use different data sets and different experiment methods. To remedy that situation, we describe below a single experimental design that combines *all* the best features of the above work. Hopefully, this design will become a template for future reports.

Like many of the researchers listed in Figure 5, we endorse holdout studies. The alternative to holdout studies is to assess the learned theory on the data used to generate it. Such non-holdout studies are useful for finding patterns in historical data. However, if the goal is to generate models that have some useful future validity, then the learned theory should be tested on data not used to build it. Failing to do so can result in a excessive over-estimate of the learned model- for example, Srinivasan and Fisher report an 0.82 correlation between the predictions generated by their learned decision tree and the actual software development effort seen in their training set [15]. However, when that data was applied to data from another project, that correlation fell to under 0.25.

We endorse holdout studies, but not jack knife studies. Witten and Frank [19] argue against jack knifing, offering examples where jack knifing can lead to widely inaccurate results. Witten and Frank's preferred alternate procedure, which is somewhat of a standard in the data mining literature, is to use holdout sets of around 10% of the available data (when the data set is large) or repeated experiments

with a 33% holdout (when the data set is small). In partial support of Witten and Frank's view, note that the Shepperd and Schofield experiments (rows $i \ldots l$) of Figure 5 and generally *lower* than, say, the COCOMO experiments (rows $\{c, m, n\}$), or Boetticher's neural net study (row p).

With the exception of Shepperd and Schofield, when researchers conduct holdout studies, they usually use large training sets and much smaller test sets; e.g. see the training sets of size 67,99,105,140 and test sets of size 10,10,10,40 used by Mair, Boetticher, Wittig, Chulani in rows $\{o, p, a, n\}$ respectively. Hence, we recommend the use of holdout studies since these can report how well a system behaves as the amount of available data is reduced.

Holdout studies should be repeated more often than (e.g.) the three runs reported by Shepperd and Schofield. After three repeats, those researchers reported convergence in their estimator at between 10 to 15 projects. But their results (Figures 1 and 2, [14, p741]) show a huge variance. We recommend repeating holdout studies 30 times in order to gain statistics on the variance.

When repeating holdout experiments, it is good practice to randomize the order of the input examples (e.g. as done by Shepperd and Schofield). Many algorithms have an *order effect* such that their performance changes dramatically if the inputs are re-ordered (the classic example is the "QuickSort" algorithm that performs badly on input that is already sorted in descending order). Kermer reported order effects in his analysis data from 15 projects: if training was restricted to 9 particular projects, the learning was far more successful [11]. Randomization avoids such order effects.

Some researchers try their calibration method on data sets used by other researchers (e.g. [13–15]), but most don't. To (partially) reduce this problem, we follow the lead of Srinivasan and Fisher and place all our data sets on the web[3] We encourage other researchers to do the same.

Experimental results should be reported using more than just mean PRED(N). Chulani et.al. report min,mean,max values seen in their holdout studies while Shepperd and Schofield offer performance graphs that let the reader see the variance in their technique. We recommend going further and reporting the results results using means *and* standard deviations seen over the 30 repeats.

It is good practice to benchmark elaborate or resource intensive techniques against simpler alternatives. In his text on empirical AI [7], Cohen argues that such comparisons are important since, sometimes, sophistication is superfluous since the simpler method can achieve similar results to the more complex method (e.g. as seen in Holte's famous study *Very Simple Classification Rules Perform Well on Most Commonly Used Datasets* [10]). We will compare our COCONUT method with a *straw man* alternative (which uses the BASE equation of Equation 1). When comparing two techniques, it is also good practice to use t-tests to compare means and deviations of two alternative techniques.

Lastly, we report results in terms of PRED(N), not MMRE. This is a pragmatic decision- we have found PRED(N) easier to explain to business users than MMRE. Also, there are more PRED(N) reports in the literature than MMRE. This is perhaps due to the influence of the COCOMO researchers who reported their 1999 landmark study using PRED(N) [4].

[3]See `http://www.vuse.vanderbilt.edu/~dfisher/tech-reports/raw-TSE-95`. and `http://promise.site.uottawa.ca/SERepository/datasets-page.html`.

```
  function learn() { # from 'D' examples
 1.   Repeats=30
 2.   while(Repeats--) {
 3.     randomizeOrderOfProjects()
 4.     for(x=2;x<=D;x += 3) {

          ###### Train on 'x' examples
 5.       for(a=2;   a <=5; a += 0.3) {
 6.         for(b=1; b<=1.3;  b += 0.05) {
 7.           sum=test(1,x,a,b,Pred)
 8.           if (sum < least) { least=sum
 9.                              BestA=a
10.                              BestB=b }}}

          ###### Test on 'D-x' examples
11.       failures = test(x+1,N,BestA,BestB,Pred)
12.       print Repeats " " i " " failures}}}

    function test(start,stop,a,b,est,act,i,mmre) {
13.   failures=0
14.   for(i=start;i<=stop;i++) {
15.     est   = a*size(i)^b*em(i)
16.     act   = actual(i)
17.     mmre  = abs((act-est)/act)*100
18.     if (mmre > pred) failures++}
19.   return failures}

     function em(i, j,out) {
20.     if   ( Base ) { return 1 }
21.     else { #return product of effort multipliers }
```

Figure 6: The COCONUT tool: pseudo-code.

6. COCONUT

This section describes COCONUT, a calibration tool that implements an incremental holdout study for the BASE model of Equation 1 and the COCOMO I model of Equation 1. COCONUT is short for "COCOMO, Not Unless Tuned".

COCONUT was inspired by Cohen's recommendation, discussed above, that complex methods should be compared to very simple ones. Figure 5 shows studies where effort estimation has been attempted using analogy, rule induction, neural nets, Bayesian tuning, regression, etc etc. To the best of our knowledge, no one has previously attempting calibration using just an exhaustive search through the range of possible a and b parameters. Such a search is tractable: just the two for-loops of lines 5/6 of the `learn` function of Figure 6.

Given D projects, the `test` function of the COCONUT tool shown in Figure 6 inputs a and b parameters and computes an effort estimate (see line 15) for the projects numbered *start* to *stop* ($start \leq stop \leq D$). The `learn` function (at lines 5 to 10) calls `test` for projects 1 to i ($i \leq D$) to find the a and b parameters that minimizes the `test` failure count. These best a and b values are then `test`ed on the projects numbered $i + 1$ to D (see lines 11,12).

COCONUT is a very simple system and could be improved. For example, Equation 1 and Equation 2 shows that a and b effect $effort$ monotonically and continuously. Hence, the exhaustive search on lines 5 and 6 of `learn` might be replaced with some form of binary search. Secondly, COCONUT is currently implemented in an interpreted C-like language (awk) and a reimplementation in "C" would make it run faster. Nevertheless, given that this sub-optimal exhaustive search implemented in an interpreted language takes just a few minutes to generate our results, we are not motivated to explore optimizations. Further, the exhaustive

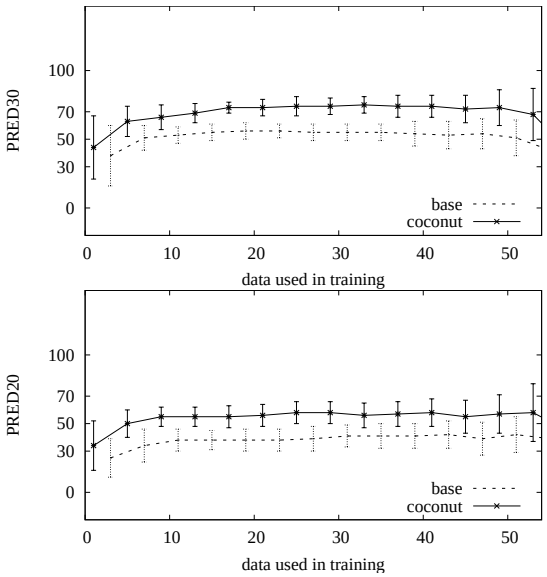

Figure 7: BASE and COCONUT on the Figure 1.

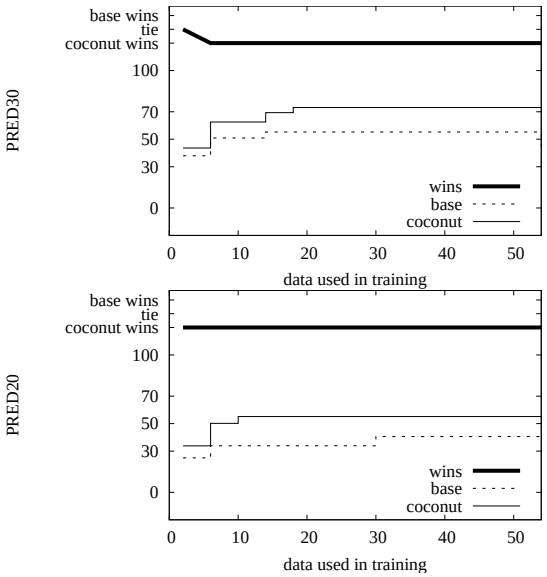

Figure 8: Hypothesis testing on the Figure 7 results.

nature of the search makes it hard to argue that some other method might do better than COCONUT.

7. INCREMENTAL HOLDOUT RESULTS

Figure 7 shows the data of Figure 1 being processed by a COCONUT incremental holdout study of the form:

$$\{Repeats = 30, |Train| = 4, |Test| = 56, \delta = 4\}|$$

That is, 30 times, the 60 projects of Figure 1 were randomly reordered and divided into a small training set (with 4 projects) and a much larger test set (with 56 items). COCONUT then calibrated COCOMO I and BASE using the training set. The best calibrations found during training for each model were then used by each model on the test set. Four items from the test site were then shifted to the training set and the training/testing process repeated. This continued until the test set was exhausted.

In Figure 7, the COCONUT-calibrating-BASE results are shown as a dashed line and the COCONUT-calibrating-COCOMO I results are shown as a solid line. Note that the mean PRED(30) values rise higher than the mean PRED(20) values since PRED(20) is a more stringent test than PRED(30).

The error bars in Figure 7 show ± 1 standard deviation around the mean for each δ is the incremental holdout study. These error bars are larger at low X values (when the training set is small) and at high X values (when the test set is small). Calibrating on a few projects, or testing on a few projects, means that quirks with those projects can disperse the results. The large variances at lower X values could explain the contradictory results reported in the introduction; i.e. when one calibration study on X=8 projects was successful but another failed.

7.1 Hypothesis Testing

7.1.1 Comparing Mean PREDs from Two Models

Figure 8 shows the results of hypothesis testing on the Figure 7 results. At the top of each plot is a thick line that can take one of three values: "base wins", "tie", and "coconut wins". This line shows the results of a *inter*-method t-test. For each X value, the null hypothesis is posed that the mean PRED values for COCONUT-calibrating-BASE is the same as for COCONUT-calibrating-COCOMO I. If a two-sided t-test accepts this hypothesis, then there is no significant difference between the PRED levels reached via calibrating the two models.

At nearly all X values, COCONUT-calibrating-COCOMO I "wins"; i.e. generates mean PRED values that are *both* significantly different (at the 95% level) to the COCONUT-calibrating-BASE results *and* which are higher than the BASE PRED mean values. To the best of our knowledge, this is the first experimental validation of the effort multipliers used in COCOMO.

7.1.2 Comparing Changes in Mean PRED From One Model

At the bottom of each plot in Figure 8 are two step functions, one for COCONUT-tuning-BASE and one for COCONUT-tuning-COCOMO I. This shows the result of the *intra*-method t-test. This t-test checks that a method's mean score at some X value is significantly different to the last significant change for that method. Such a comparison could reject the hypothesis that two adjacent and different means are truly different, if the variances of the means were too large.

When the mean performance of the studies methods stops changing significantly, then methods have *plateau-ed* and further data collection for that learner is superfluous. At the 95% confidence level, in Figure 8, PRED(20) plateaus at 54and PRED(30) plateaus at 72% after X=12 and X=20 projects respectively. As might be expected, it takes longer to achieve good results on the more stringent PRED(20) criteria than the weaker PRED(30) criteria.

8. DISCUSSION

A PRED(30)=70% result on holdout data seems as good as the state-of-the-art results shown in Figure 5. For ex-

ample, the COCONUT result seems very close to the COCOMO II results of PRED(30) of 69% on holdout data reported by Chulani et.al. [4]. Further, COCONUT achieves this level of performance using *far less effort* that the Chulani et.al. result:

- Chulani et.al.'s improvements on COCOMO-I needed 78 more projects, a DELPHI panel of experts, and some Bayesian calibration.
- The peak PRED(30) values in Figure 7 were generated using the COCOMO I effort multipliers, 40 more projects, no DELPHI panel, no Bayesian calibration, and the very simple exhaustive enumeration technique of COCONUT.

However, directly comparing the Figure 8 results with Figure 5 is problematic. Chulani et.al. did not report the standard deviation in their holdout experiments and without that information, it is difficult to compare our method to theirs. To be fair, it is quite correct that they did not report standard deviation since they repeated their holdouts only 15 times (a number too small to collect accurate information about standard deviations). Nevertheless, should they ever repeat their study using 20 to 30 repeats, it would be interesting to view their standard deviation results.

Also, the Chulani and COCONUT results come from different dataset (161 records in COCOMO II format vs 60 NASA records in COCOMO I format) and it would be preferred to apply the same techniques to same dataset (something we hope to do in the near future).

9. THREATS TO EXTERNAL VALIDITY

Two threats to our study come from *linearity* and *granularity* of the effort models. If the effort model is non-linear, then there could exist some region in the space of the calibration parameters $< a, b >$ where the performance of the estimation model changes radically. Linearity is a problem in any extrapolation-based method that assumes that the region between two samples changes smoothly between the two samples. Granularity is a related concern: if a model is searched in steps of size M, and the model's behavior can change dramatically is some space $N \leq M$, then the explorer can miss important features of the model.

COCONUT's complete search through the range of $< a, b >$ values makes no extrapolation assumptions. Hence, COCONUT is immune to linearity problems.

Theoretically, COCONUT could suffer from the granularity problem. However, Figure 6 steps through a in steps of 0.3 and b in steps of 0.05 and these steps are smaller than the differences we see in the literature for $< a, b >$ calibrations for COCOMO I. Hence, we are somewhat confident that COCONUT does not have a granularity problem.

Other threats to external validity are covered by the incremental holdout study methodology. By randomizing the order of the input examples, we are reducing the impact of order effects. By conducting holdout studies, we are reducing the chance of over-fitting. By only reported significant changes (found via hypothesis testing), we are reducing the odds that our results arise from some statistical quirk.

Nevertheless, the results presented in this paper come from a single dataset. and, clearly, the above analysis needs to be repeated on other data sets. Currently, we are exploring access to COCOMO II data.

10. FUTURE WORK

Our results suggest an alternative to the Chulani et.al. method for using expert opinion to calibration COCOMO-like models.

One of the main motivations for the Bayesian analysis of COCOMO-II was that the regression results from the 83+78 projects had slopes that contradicted certain expert intuitions. For example, regression of the COCOMO data concluded that building reusable components *decreased* development costs. Most experts believe that the extra effort required to generalize a design actually *increases* the cost of building such components. This anomaly was explained as follows: the 83+78 projects did not contain enough samples of projects that make heavy use of reuse. To DELPHI panel and the subsequent Bayesian calibration was used to fill in the gaps in the project data with expert knowledge. This combination of DELPHI+Bayesian methods proved successful: COCOMO-II had much higher PRED(N) levels than COCOMO-I.

If 20 projects is enough for local calibration, then it might be possible to simplify the DELPHI tuning process used by Chulani et.al. in the COCOMO II project. Descriptions of 20 projects could be generated from, say, 4 experts each asked to describe 5 projects representing the kind of work conducted by their company. Unlike a DELPHI panel, this group of experts would not need to conduct extensive discussions to explore their conclusions. If those descriptions were made in terms of the COCOMO-I parameters, COCONUT could then tune an effort model to that expert option using those 20 expert-generated examples. We acknowledge that this proposal is speculative. Nevertheless, if successful, it could significantly reduce the data collection effort required to calibrate a local model.

11. CONCLUSION

COCONUT conducts an exhaustive search over the space of the $< a, b >$ parameters in a COCOMO I model. This technique is much simpler than other methods used to learn good effort models including neural nets [3,13,15,20], linear regression [1,2,13,14], rule induction [13], DELPHI/Bayesian tuning [2, 4], and analogical reasoning [14], Further COCONUT seems to yield PRED levels comparable to other methods, and does so with less project data and fewer attributes (no scale factors).

COCONUT was analyzed here using incremental holdout (not jack knife) studies, combined with randomization and hypothesis testing, repeated 30 times. We commend this experimental method to the effort estimation community since it controls for the variance problem in holdout experiments; and can report the minimum point at which enough project data is enough.

This paper is offered as a baseline result with the hope that other researchers will try to reproduce and out-perform our results. Using incremental holdout studies, it would be very clear if some effort estimation method out-perform COCONUT. Such an alternate technique would be superior if (a) it was simpler to implement or faster to execute than COCONUT; (b) it achieves higher predictive accuracies or lower variances than Figure 7 or Figure 8; and (c) it do so using less project data than COCONUT. To facilitate the search for a better-than-COCONUT technique, we have followed the lead of Srinivasan and Fisher, and placed all

our data on the web.

We would also encourage the creation of public domain effort estimation data sets. Naturally, we would have preferred to have conduct this study using other data sets such as COCOMO-II. However, methodologically, it is important that our results be reproducible. Therefore, we used the COCOMO-I data in Figure 1 and look forward to the time when we can try conduct incremental holdout studies with COCONUT on other data sets

Finally, the COCOMO II rule that "10 projects is enough for local calibration" needs some qualification. Certainly, Figure 8 shows that improvements in mean PRED start leveling off after 5-10 projects. However, at 5-10 projects, the variance in the mean PRED values is still decreasing and we we have seen statistically significant PRED improvements up to 20 projects.

Acknowledgments

This research was conducted at Portland State University under NASA contract NCC2-0979 and NCC5-685. The work was sponsored 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.

12. REFERENCES

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

[2] 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.

[3] G. Boetticher. When will it be done? the 300 billion dollar question, machine learner answers. *IEEE Intelligent Systems*, June 2003.

[4] 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.

[5] S. Chulani, B. Boehm, and B. Steece. From multiple regression to bayesian analysis for calibrating COCOMO II. *Journal of Parametrics*, 15(2):175–188, 1999.

[6] B. Clark. *The Effects of Process Maturity on Software Development Effort*. PhD thesis, University of Southern California, 1997. Available from `http://sunset.usc.edu/~bkclark/Research/PMAT990406.pdf`.

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

[8] D. Ferens and D. Christensen. Calibrating software cost models to Department of Defense Database: A review of ten studies. *Journal of Parametrics*, 18(1):55–74, November 1998.

[9] H. Habib-agahi, S. Malhotra, and J. Quirk. Estimating software productivity and cost for NASA projects. *Journal of Parametrics*, pages 59–71, November 1998.

[10] R.C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11:63, 1993.

[11] C.F. Kemerer. An empirical validation of software cost estimation models. *Communications of the ACM*, 30(5):416–429, May 1987.

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

[13] Carolyn Mair, Gada Kadoda, Martin Lefley, Keith Phalp, Chris Sch ofield1, Martin Shepperd, and Steve Webster. An investigation of machine learning based prediction systems. *The Journal of Systems and Software*, 53(1):23–29, 2000.

[14] M. Shepperd and C. Schofield. Estimating software project effort using analogies. *IEEE Transactions on Software Engineering*, 23(12), November 1997. Available from `http://www.utdallas.edu/~rbanker/SE_XII.pdf`.

[15] K. Srinivasan and D. Fisher. Machine learning approaches to estimating software development effort. *IEEE Trans. Soft. Eng.*, pages 126–137, February 1995.

[16] The Standish Group Report: Chaos, 1995. Available from `http://www4.in.tum.de/lehre/vorlesungen/vse/WS2004/1995_Standish_Chaos.%pdf`.

[17] S. Stukes and H. Apgar. Applications oriented software data collection: Software model calibration report,TR-9007/549-1, management consulting and research, 15, March 1991.

[18] S. Stukes and D. Ferens. Software cost model calibration. *Journal of Parametrics*, 18(1):77–98, 1998.

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

[20] G. Wittig and G. Finnie. Estimating software development effort with connectionist models. *Information and Software Technology*, 39(7):469–476, 1997.