40 Years of Parametric Effort Estimation: A Report Card on COCOMO-style Research

Tim Menzies*, Barry Boehm†, Ye Yang‡, Jairus Hihn§, and Naveen Lekkalapudi¶
*Computer Science, North Carolina State University, USA, tim.menzies@gmail.com
†Computer Science, University Southern California, USA, barryboehm@gmail.com
‡Systems and Enterprise, Stevens Institute of Technology, USA, yangye@gmail.com
§Jet Propulsion Laboratory/California Institute of Technology, USA, jairus.hihn@jpl.nasa.gov
¶Computer Science, West Virginia University, USA, nalekkalapudi@mix.wvu.edu

***Abstract*—The longevity of parametric effort estimation is remarkable. Decades after their invention, these methods are still both widely used and widely useful.**

This paper reviews the standard criticisms of this approach. We find that, contrary to common criticisms, (1) parametric estimation has not been superseded by more recent estimation methods; (2) it is not true that parametric estimation is no better than simplistic lines of code counts; (3) the old parametric calibration data is still relevant to more recent projects; (4) parametric estimation need not be expensive to deploy at some new site since these these methods can be tuned on very small sample sizes (in our experiments, a mere eight projects is enough); and (5) compared to other methods, parametric estimation is not unduly sensitive to errors in the size estimates.

Hence we conclude that, in 2015, is still valid and recommended practice to try parametric estimation before exploring other, more innovative methods. Also, it can be useful to augment parametric estimation with (a) some local calibration and (b) some column pruning (examples of those techniques are discussed in this paper).

I. Introduction

Accurately estimating software development effort is of vital importance. Under-estimation leads to schedule and budget overruns and, perhaps, project cancellation [59]. Over-estimation slows the deployment of promising ideas and organizational competitiveness [29]. It is hence not surprising that there is long history of researchers exploring software effort estimation, dating back to the 1970s[1]. In 2007, Jorgensen and Shepperd reported on hundreds of research papers devoted to the topic, over half of which propose some innovation for developing new estimation models [20]. Since then, many more such papers have been published[2].

In the 1970s and 1980s, this kind of research was focused on *parametric estimation* as done by Putnam and others [5], [6], [16], [17], [61], [62]. For example, Boehm's COnstructive COst MOdel (COCOMO) model [6] assumes that effort varies exponentially on size as seen in this parametric form: $effort \propto a \times KLOC^b$. To deploy this equation in an organization, local project data is used to tune the (a, b) parameter values, If local data is unavailable, new projects can reuse prior tunings, perhaps with some minor tweaks [44].

Since the early work on parametric estimation, other researchers have innovated other estimation methods based on regression trees [57] case-based-reasoning [57], spectral clustering [42], genetic algorithms [8], [12], etc. These methods can be augmented with "meta-level" techniques like tabu search [11], feature selection [10], instance selection [28], feature synthesis [45], active learning [30], transfer learning [31]. temporal learning [40], [47], and many more besides.

In her keynote address to ICSE'01, Mary Shaw [56] noted that it can take up to a decade for research innovations to become stable and then another decade after that to become widely popular[3]. Given that, it would be reasonable to expect commercial adoption of the 1990s estimation work on regression trees [57] or case-based-reasoning [57]. This has not happened:

- Based on our knowledge of the Chinese and United States software industry, we assert that industry, government labs, or professional societies make almost exclusive use parametric estimation tools such as those offered by Price Systems (pricesystems.com) and Galorath (galorath.com).
- Parametric estimation is widely-used, especially across the aerospace industry and various U.S. government agencies. For example, NASA routinely checks software estimates in COCOMO [13].
- Professional societies, handbooks and certification programs have developed around the use of these parametric estimation methods and tools such as ICEAA (the International Cost Estimation and Analysis Society: iceaaonline.com), along with workshops such as the NASA Cost Symposium (goo.gl/8jxPrb) and the International Forum on COCOMO and Systems/Software Cost Modeling (goo.gl/O01Pc6).

[1]e.g. [5], [6], [8], [16]–[18], [20], [43], [52], [55], [57], [60]–[62]
[2]e.g. [11], [23]–[25], [28], [30], [31], [34], [38], [39], [45], [47]
[3]See slide3 of her slides at goo.gl/tfi3D6.

To explain this puzzling lack of adoption of more innovative effort estimation methods, we make the following conjecture: the case has not been made that #1 parametric estimation *is no longer useful* and that #2 the supposedly more innovation methods are *comparatively more useful*. Since the case for #1 and #2 is missing in the literature, we choose to explore these two issues.

For that task, we used COCOMO since its internal details have been fully published [7]. Also, several COCOMO data sets are readily available which we used to answer five research questions:

RQ1: Is parametric estimation no better than just using simple Lines of Code measures? This is an often heard, but rarely tested, criticism.

RQ2: Has parametric estimation been superseded by more recent estimation methods? We apply our "best-of-breed" learner, published in recent IEEE TSE paper, to some recent NASA data. We also comparatively assess case-based reasoning and regression trees.

RQ3: Are the old parametric tunings irrelevant to more recent projects? We apply the old COCOMO-II tunings from 2000, with and without local tweaking, to a wide range of projects dating from 1970 to 2010.

RQ4: Is parametric estimation expensive to deploy at some new site? We experiment with tuning estimation models on very small training sets as well as simplifying the specification of local projects.

RQ5: Are parametric estimates unduly sensitive to errors in the size estimate? In the context of RQ4, when we are using very small training sets, we check what happens if there are large errors in the KLOC estimate.

The answer to all of these research questions was "no"; i.e. this paper fails to prove #1 and #2. Instead, we showed that the continued use of parametric estimation can still be endorsed.

We believe this result has strong implications for current practice in effort estimation research. Specifically: it must be asked to what extent we can count on newer innovative techniques for effort estimation. Small training sets are very common in software estimation studies[4]. This small quantity, as well as the unique and highly variable characteristics of SE project data place great limitation on the results obtained by naively applying some brand-new algorithm. What COCOMO, or other parametric estimation models, can offer largely contains the backbone structure for making estimation decisions from expert-delphi, as well as well calibrated tuning factors from over 40 years of industrial data. Perhaps the best future direction is to investigate how innovative new techniques can extend (rather than replace) existing and successful estimation methods. Some experiments in this paper are a starting point for that investigation.

[4]For example, five recent effort estimation publications [3], [4], [29], [37], [41] used tables of data with 13,15,31,33,52 rows (respectively).

II. Background

A. Delpi- vs Model-based Estimation

There are two broad classes of software effort estimation: *delphi-based methods* use human experts while *model-based methods* such as parametric estimation (a) build a model using old data then (b) use the model to generate estimates for new projects.

Once a large software project has been divided into multiple small stories, Delphi-based methods can be remarkably accurate [49]. However, for *initial* large-scale estimates, the Delphi approach has some problems. Passos et al. show that many commercial software engineers generalize from their first few projects to all future projects [53]. Also, Jorgensen & Gruschke document how commercial estimation "gurus" rarely use lessons from past projects to improve their future estimates [19]. When engineers fail to revise their beliefs, this leads to poor Delphi-based estimates (see examples in [19]).

B. COCOMO: Origins and Development

These concerns with Delphi date back many decades and were the genesis for COCOMO. In 1976, Robert Walquist (a TRW division general manager) told Boehm:

> *"Over the last three weeks, I've had to sign proposals that committed us to budgets of over $50 million to develop the software. In each case, nobody had a good explanation for why the cost was $50M vs. $30M or $100M, but the estimates were the consensus of the best available experts on the proposal team. We need to do better. Feel free to call on experts & projects with data on previous software cost."*

TRW had a previous model that worked well for a part of TRWs software business [62], but it did not relate well to the full range of embedded software, command and control software, and engineering and scientific software involved in TRWs business base. Having access to experts and data was a rare opportunity, and a team involving Ray Wolverton, Kurt Fischer, and Boehm conducted a series of meetings and Delphi exercises to determine the relative significance of various software cost drivers. Using combining local expertise and data, plus some prior results such as [5], [17], [55], [61], and early versions of the RCA PRICE S model [16], a model called SCEP was created (Software Cost Estimation Program). Except for one explainable outlier, the models estimates for the 20 projects with solid data were within 30% of the actuals, with most within 15% of the actuals.

After gathering some further data from subsequent TRW projects and about 35 projects from teaching software engineering courses at UCLA and USC along with commercial short courses on software cost estimation, Boehm was able to gather 63 data points that could be

	Definition	Low-end = {1,2}	Medium ={3,4}	High-end= {5,6}
Scale factors:				
Flex	development flexibility	development process rigorously defined	some guidelines, which can be relaxed	only general goals defined
Pmat	process maturity	CMM level 1	CMM level 3	CMM level 5
Prec	precedentedness	we have never built this kind of software before	somewhat new	thoroughly familiar
Resl	architecture or risk resolution	few interfaces defined or few risks eliminated	most interfaces defined or most risks eliminated	all interfaces defined or all risks eliminated
Team	team cohesion	very difficult interactions	basically co-operative	seamless interactions
Effort multipliers				
acap	analyst capability	worst 35%	35% - 90%	best 10%
aexp	applications experience	2 months	1 year	6 years
cplx	product complexity	e.g. simple read/write statements	e.g. use of simple interface widgets	e.g. performance-critical embedded systems
data	database size (DB bytes/SLOC)	10	100	1000
docu	documentation	many life-cycle phases not documented		extensive reporting for each life-cycle phase
ltex	language and tool-set experience	2 months	1 year	6 years
pcap	programmer capability	worst 15%	55%	best 10%
pcon	personnel continuity (% turnover per year)	48%	12%	3%
plex	platform experience	2 months	1 year	6 years
pvol	platform volatility ($\frac{frequency\ of\ major\ changes}{frequency\ of\ minor\ changes}$)	$\frac{12\ months}{1\ month}$	$\frac{6\ months}{2\ weeks}$	$\frac{2\ weeks}{2\ days}$
rely	required reliability	errors are slight inconvenience	errors are easily recoverable	errors can risk human life
ruse	required reuse	none	multiple program	multiple product lines
sced	dictated development schedule	deadlines moved to 75% of the original estimate	no change	deadlines moved back to 160% of original estimate
site	multi-site development	some contact: phone, mail	some email	interactive multi-media
stor	required % of available RAM	N/A	50%	95%
time	required % of available CPU	N/A	50%	95%
tool	use of software tools	edit,code,debug		integrated with life cycle

Fig. 1. COCOMO-II attributes.

published and to extend the model to include alternative development modes that covered other types of software such as business data processing. The resulting model was called the COnstructive COst MOdel, or COCOMO, and was published along with the data in the book Software Engineering Economics [6]. In COCOMO-I, project attributes were scored using just a few coarse-grained values (very low, low, nominal, high, very high). These attributes are *effort multipliers* where a off-nominal value changes the estimate by some number greater or smaller than one. In COCOMO-I, all attributes (except KLOC) influence effort in a linear manner.

Following the release of COCOMO-I Boehm created a consortium for industrial organizations using COCOMO . The consortium collected information on 161 projects from commercial, aerospace, government, and non-profit organizations. Based on an analysis of those 161 projects, Boehm added new attributes called *scale factors* that had an *exponential impact* on effort (e.g. one such attribute was process maturity). For a full description of the COCOMO-II attributes, see Figure 1.

Using that new data, Boehm and his colleagues developed the *tunings* shown in Figure 2 that map the project descriptors (very low, low, etc) into the specific values used in the COCOMO-II model (released in 2000 [7]):

$$effort = a \prod_i EM_i * KLOC^{b+0.01 \sum_j SF_j} \quad (1)$$

Here, *EM,SF* denote the effort multipliers and scale factors and a, b are the *local calibration* parameters (which in COCOMO-II have default values of 2.94 and 0.91). Also, *effort* measures "development months" where one month is 152 hours work by one developer (and includes development and management hours). For example, if *effort*=100, then according to COCOMO, five developers would finish the project in 20 months.

Recent changes in the software industry suggest it is time to revise COCOMO-II. The rise of agile methods, web services, cloud services, parallelized software on multicore chips, field-programmable-gate-array (FPGA) software, apps, widgets, and net-centric systems of systems (NCSOS) have caused the COCOMO II developers and users to begin addressing an upgrade to the 14-year-old COCOMO II. Current discussions of a potential COCOMO III have led to a reconsideration of the old COCOMO 1981 development modes, as different development phenomena appear to drive the costs and schedules of web-services, business data processing, real-time embedded software, command and control, and engineering and scientific applications[5].

[5]Efforts to characterize these modes and to gather data to calibrate models for dealing with them are underway, and contributors to the definition and calibration are welcome.

```
_  = None;  Coc2tunings = [[
#               vlow  low   nom   high  vhigh  xhigh
# scale factors:
'Flex',        5.07, 4.05, 3.04, 2.03, 1.01,     _],[
'Pmat',        7.80, 6.24, 4.68, 3.12, 1.56,     _],[
'Prec',        6.20, 4.96, 3.72, 2.48, 1.24,     _],[
'Resl',        7.07, 5.65, 4.24, 2.83, 1.41,     _],[
'Team',        5.48, 4.38, 3.29, 2.19, 1.01,     _],[
# effort multipliers:
'acap',        1.42, 1.19, 1.00, 0.85, 0.71,     _],[
'aexp',        1.22, 1.10, 1.00, 0.88, 0.81,     _],[
'cplx',        0.73, 0.87, 1.00, 1.17, 1.34, 1.74],[
'data',           _, 0.90, 1.00, 1.14, 1.28,     _],[
'docu',        0.81, 0.91, 1.00, 1.11, 1.23,     _],[
'ltex',        1.20, 1.09, 1.00, 0.91, 0.84,     _],[
'pcap',        1.34, 1.15, 1.00, 0.88, 0.76,     _],[
'pcon',        1.29, 1.12, 1.00, 0.90, 0.81,     _],[
'plex',        1.19, 1.09, 1.00, 0.91, 0.85,     _],[
'pvol',           _, 0.87, 1.00, 1.15, 1.30,     _],[
'rely',        0.82, 0.92, 1.00, 1.10, 1.26,     _],[
'ruse',           _, 0.95, 1.00, 1.07, 1.15, 1.24],[
'sced',        1.43, 1.14, 1.00, 1.00, 1.00,     _],[
'site',        1.22, 1.09, 1.00, 0.93, 0.86, 0.80],[
'stor',           _,    _, 1.00, 1.05, 1.17, 1.46],[
'time',           _,    _, 1.00, 1.11, 1.29, 1.63],[
'tool',        1.17, 1.09, 1.00, 0.90, 0.78,     _]]

def COCOMO2(project,
            a = 2.94, b = 0.91, # defaults
            tunes= Coc2tunings):# defaults, see above
  sfs  = 0
  ems  = 1
  kloc = 22
  scaleFactors = 5
  effortMultipliers = 17
  for i in range(scaleFactors):
    sfs += tunes[i][project[i]]
  for i in range(effortMultipliers):
    j = i + scaleFactors
    ems *= tunes[j][project[j]]
  return a * ems * project[kloc] ** (b + 0.01*sfs)
```

Fig. 2. COCOMO-II: effort estimates from a *project*. Here, *project* has up to 24 attributes (5 scale factors plus 17 effort multipliers plus KLOC plus. in the training data, the actual effort). Each attribute except KLOC and effort is scored using the scale very low = 1, low=2, etc. For an explanation of the attributes shown in green, see Figure 1.

C. COCOMO and Local Calibration

When there is insufficient data to generate the exact tunings of Figure 2, a *local calibration* procedure can be used. Local calibration adjusts the impact of the scale factors and effort multipliers by tuning the a, b values of Equation 1 (while keeping constant the other values of the tuning matrix shown in Figure 2).

Menzies' preferred local calibration procedure is the COCONUT procedure of Figure 3 (first written in 2002 and first published in 2005 [44]). For some number of *repeats*, COCONUT will *ASSESS* some *GUESSES* for (a, b) by applying them to some *training* data. If any of these guesses prove to be *useful* (i.e. reduce the estimation error) then COCONUT will recurse after *constricting* the guess range for (a, b) by some amount (say, by $2/3$rds). COCONUT terminates when (a) nothing better is found at the current level of recursion or (b) after 10 recursive calls– at which point the guess range has been constricted to $(2/3)^{10} \approx 1\%$ of the initial range.

```
def COCONUT(training,          # list of projects
            a=10, b=1,         # initial  (a,b) guess
            deltaA    = 10,    # range of "a" guesses
            deltaB    = 0.5,   # range of "b" guesses
            depth     = 10     # max recursive calls
            constricting=0.66):# next time,guess less
  if depth > 0:
    useful,a1,b1= GUESSES(training,a,b,deltaA,deltaB)
    if useful: # only continue if something useful
      return COCONUT(training,
                     a1, b1,  # our new next guess
                     deltaA * constricting,
                     deltaB * constricting,
                     depth - 1)
  return a,b

def GUESSES(training, a,b, deltaA, deltaB,
            repeats=20): # number of guesses
  useful, a1,b1,least,n = False, a,b, 10**32, 0
  while n < repeats:
    n += 1
    aGuess = a - deltaA + 2 * deltaA * rand()
    bGuess = b - deltaB + 2 * deltaB * rand()
    error  = ASSESS(training, aGuess, bGuess)
    if error < least: # found a new best guess
      useful,a1,b1,least = True,aGuess,bGuess,error
  return useful,a1,b1

def ASSESS(training, aGuess, bGuess):
   error = 0.0
   for project in training: # find error on training
     predicted = COCOMO2(project, aGuess, bGuess)
     actual    = effort(project)
     error    += abs(predicted - actual) / actual
   return error / len(training) # mean training error
```

Fig. 3. COCONUT: tuning the a, b local calibration variables used in COCOMO's effort equation of $a * \prod EM * KLOC^{b+0.01 \sum SF}$. Uses the COCOMO2 function of Figure 2.

Types of projects	COC81	NASA93	COC05	NASA10
Avionics		26	10	17
Banking			13	
Business apps/data processing	7	4	31	
Control	9	18	13	
Human-machine interaction	12			
Military, ground			8	
Misc	5	4	5	
Mission Planning		16		
SCI scientific application	16	21	11	
Support (tools, utilities, etc)	7			
Sys (OS, compilers, sensors,etc)	7	3	2	

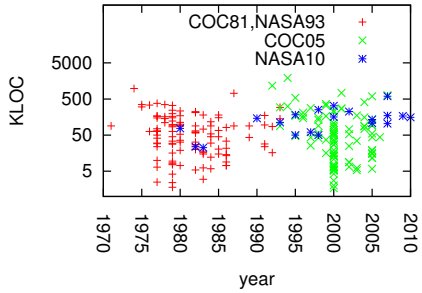

Fig. 4. Projects in this study. Figure 1 shows project attributes.

III. Experimental Methods

This section offers some notes on the experimental methods used later in this paper.

A. Choice of Data

For this study we used all the COCOMO data available to the authors (see Figure 4). The COC81 and NASA93 data comes from the PROMISE repository[6]:

- COC81 is the original data from 1981 COCOMO book;
- NASA93 is data collected at NASA in the early 1990s about software that supported the planning activities for the International Space Station.

The other data sets come from more recent data collections by the COCOMO team (COC05) or by NASA's effort estimation team (NASA10). These COC05 and NASA10 data sets are proprietary and, as such, cannot be released to the rest of the research community. For information on the kinds of projects included in these two data sets, see Figure 4.

As shown in Figure 4, the data tables of NASA10, COC81, NASA93, and COC05 contain information on 17, 63, 93, and 95 projects, respectively. As to the attributes of that data, all the projects contain the COCOMO attributes of Figure 1.

B. Choice of Experimental Rig

"Ecology inference" is the conceit that what holds for all, also holds for parts of the population [42], [54]. For the purposes of avoid ecological inference, our rig runs separately for each data set (see Figure 5).

For the purposes of repeatability, we use leave-one-out experiments i.e. for every project, the learner is offered a training set containing all other projects. For a justification of leave-one-out vs other methods (e.g. n-way cross-validation), see [27].

Since some of our methods include a stochastic algorithm (the COCONUT algorithm of Figure 3), we repeat the leave-one-out experiments $N = 10$ times (10 was selected since, after experimentation, we found our results looked the same at $N = 8$ and $N = 16$).

The performance of the estimate for each project was assessed via the magnitude of the relative error; i.e. $MRE = \frac{abs(actual - predicted)}{actual}$. Shepperd & MacConnell [58] propose another measure that reports the performance as a ratio of some other, much simpler, "straw man" approach (they recommend the mean effort value of $N > 100$ random samples of the training data). At first, we used the Shepperd & MacDonnel approach for this work but found that their straw man had orders of magnitude larger error than all the results shown here. Hence, we adopt the spirit, but not the letter, of their proposal and compare all our results against the LOC(n) "straw man" method discussed below.

[6]https://promisedata.googlecode.com/svn/trunk/effort/

```
def RIG():
 DATA = { COC81, NASA83, COC05, NASA10 }
 for data in DATA # e.g. data = COC81
     mres= {}
     for learner in LEARNERS # e.g. learner = COCONUT
       n = 0
       10 times repeat:
         for project in DATA #  e.g.  one project
           training = data - project # leave-one-out
           model    = learn(training)
           estimate = guess(model, project)
           actual   = effort(project)
           mre      = abs(actual - estimate)/actual
           mres[learner][n++] = mre
     print rank(mres) # some statistical tests
```

Fig. 5. The experimental rig used in this paper.

C. Choice of Learners

Our LOC(n) "straw man" estimators ignore all attributes except for lines of code in the n nearest projects[7]. For $n > 1$, we weight estimates via the triangle function of Walkerden and Jeffery [60]; e.g.. for $loc(3)$, the estimate from the first, second, third closest neighbor with estimates a, b, c is

$$effort = (50a + 33b + 17c)/100 \quad (2)$$

Apart from the LOC "straw man", we also compare COCOMO-II and COCONUT with CART and Knear(n). CART is an *iterative dichotomization* algorithm that finds the attribute that most divides the data such that the variance of the goal variable in each division is minimized. The algorithm then recurses on each division. Finally, the cost data in the leaf divisions is averaged to generate the estimate.

Knear(n) estimates a new project's effort by a nearest neighbor method [57]. Unlike LOC(n), a Knear(n) method uses all attributes (all scale factors and effort multipliers as well as lines of code) to find the *n-th* nearest projects in the training data. Knear(3) combines efforts from three nearest neighbors using Equation 2. Knear(n) is an example of CBR; i.e. *case-based reasoning*. CBR for effort estimation was first pioneered by Shepperd & Schofield in 1997 [57]. Since then, it has been used extensively in software effort estimation [3], [21], [23]–[26], [33]–[35], [38], [57], [60]. There are several reasons for this. Firstly, it works even if the domain data is sparse [50]. Secondly, unlike other predictors, it makes no assumptions about data distributions or some underlying parametric model.

In defense of our selection of CART and case-based reasoners like Knear(n):

[7]For distance, we use the standard Euclidean measure recommended for instance-based reasoning by Aha et al. [1]; i.e. $\sqrt{\sum_i (x_i - y_i)^2}$ where x_i, y_i are values normalized 0..1 for the range min..max

Recent studies published in IEEE TSE [14], [25], [30] endorse CART as a state-of-the-art effort estimation method. These studies assert that in terms of assessing new effort estimation methods, existing methods such as CART's regression tree generation may be more than adequate, e.g. Dejaeger et al. found little evidence that learners more elaborate than CART offer a significant value-added [14]. Our own results endorse the conclusions of Dejaeger et al. Recently we studied 90 effort estimators. These were combinations of 10 pre-processors and 9 learners to build ensemble methods [29]). As it might have been predicted by Myrtveit et al. [51], the ranking of the estimators varied greatly. However, we found a small group of 13 estimators (all variants of CART and Knear(n) aided with pre-processors) that were consistently the best performing methods across a range of experimental rigs.

Fig. 6. Studies concluding that CART and Kmeans(n) are useful as comparison methods in effort estimation.

- Recall Shaw's comments listed in the introduction. If it takes decades before research work becomes popular in industry, then it makes most sense for this study to explore innovative effort estimation methods that at least a decade old, or more. Two such methods are CART and Knear(n).
- Several papers have concluded that CART and Knear(n) are useful comparison algorithms for effort estimation– see Figure 6.

D. Choice of Statistical Ranking Methods

The last line of our experimental rig shown in Figure 5 *rank*s multiple methods for learning effort estimators. This study ranks methods using the Scott-Knott procedure recommended by Mittas & Angelis in their 2013 IEEE TSE paper [48]. This method sorts a list of l treatments with ls measurements by their median score. It then splits l into sub-lists m, n in order to maximize the expected value of differences in the observed performances before and after divisions. E.g. for lists l, m, n of size ls, ms, ns where $l = m \cup n$:

$$E(\Delta) = \frac{ms}{ls} abs(m.\mu - l.\mu)^2 + \frac{ns}{ls} abs(n.\mu - l.\mu)^2$$

Scott-Knott then applies some statistical hypothesis test H to check if m, n are significantly different. If so, Scott-Knott then recurses on each division. For example, consider the following data collected under different treatments *rx*:

```
rx1 = [0.34, 0.49, 0.51, 0.6]
rx2 = [0.6,  0.7,  0.8,  0.9]
rx3 = [0.15, 0.25, 0.4,  0.35]
rx4=  [0.6,  0.7,  0.8,  0.9]
rx5=  [0.1,  0.2,  0.3,  0.4]
```

After sorting and division, Scott-Knott declares:

- Ranked #1 is rx5 with median= 0.25
- Ranked #1 is rx3 with median= 0.3
- Ranked #2 is rx1 with median= 0.5
- Ranked #3 is rx2 with median= 0.75
- Ranked #3 is rx4 with median= 0.75

Note that Scott-Knott found little difference between rx5 and rx3. Hence, they have the same rank, even though their medians differ.

Scott-Knott is preferred to, say, hypothesis testing over all-pairs of methods[8]. To avoid an all-pairs comparison, Scott-Knott only calls on hypothesis tests *after* it has found splits that maximize the perfromance differences.

For this study, our hypothesis test H was a conjunction of the A12 effect size test of and non-parametric bootstrap sampling; i.e. our Scott-Knott divided the data if *both* bootstrapping and an effect size test agreed that the division was statistically significant (99% confidence) and not a "small" effect ($A12 \geq 0.6$).

For a justification of the use of non-parametric bootstrapping, see Efron & Tibshirani [15, p220-223]. For a justification of the use of effect size tests see Shepperd&MacDonell [58]; Arcuri&Briand [2]; Kampenes [22]. These researchers warn that even if an hypothesis test declares two populations to be "significantly" different, then that result is misleading if the "effect size" is very small[9]. Hence, to assess the performance differences we first must rule out small effects. Vargha and Delaney's non-parametric A12 effect size test explores two lists M and N of size m and n:

$$A12 = \left(\sum_{x \in M, y \in N} \begin{cases} 1 & if\ x > y \\ 0.5 & if\ x == y \end{cases} \right) / (mn)$$

This expression computes the probability that numbers in one sample are bigger than in another. This test was recently endorsed by Arcuri and Briand at ICSE'11 [2].

IV. Results

A. COCOMO vs Just Lines of Code

This section explores **RQ1: is parametric estimation no better than using simple lines of code measures?**

An often heard, but not often tested, criticism of parametric estimation methods is that they are no better than just using simple lines of code measures. As shown in Figure 7, this is not necessarily true. This figure is a comparative ranking for LOC(1) LOC(3), COCOMO-II and COCONUT. The rows of Figure 7 are sorted by the median MRE figure. These rows are divided according to their *rank*, shown in the left column: better methods have *lower rank* since they have *lower MRE* error values. The right-hand-side column displays the median error (as a black dot) inside the inter-quartile range (25th to 75th percentile, show as a horizontal line).

[8] e.g. Six treatments can be compared $(6^2 - 6)/2 = 15$ ways. A 95% confidence test run 15 times total confidence $0.95^{15} = 46\%$.

[9] For example, Kocaguenli et al. [32] report on the misleading results of such hypothesis tests in software defect prediction (due to small size of the effect being explored).

NASA10 (new NASA data up to 2010):

rank	treatment	median	IQR
1	COCOMO-II	42	35
2	COCONUT	47	34
3	loc(3)	49	97
3	loc(1)	67	44

COC05 (new COCOMO data up to 2005):

rank	treatment	median	IQR
1	COCOMO-II	46	146
1	loc(1)	55	114
1	loc(3)	65	99
1	COCONUT	65	34

NASA93 (NASA data up to 1993):

rank	treatment	median	IQR
1	COCONUT	35	38
1	COCOMO-II	38	39
2	loc(1)	62	54
2	loc(3)	75	102

COC81 (original data from the 1981 COCOMO book):

rank	treatment	median	IQR
1	COCOMO-II	33	35
1	COCONUT	37	42
2	loc(3)	80	237
2	loc(1)	84	100

Fig. 7. COCOMO vs just lines of code. MRE values seen in leave-one-studies, repeated ten times. For each of the four tables in this figure, *better* methods appear *higher* in the tables. In these tables, median and IQR are the 50th and the (75-25)th percentiles. The IQR range is shown in the right column with black dot at the median. Horizontal lines divide the "ranks" found by Scott-Knott (shown in left column).

NASA10: (new NASA data up to 2010):

rank	treatment	median	IQR
1	COCOMO-II	42	35
2	COCONUT	46	33
3	Knear(3)	50	77
3	Knear(1)	57	49
3	CART	61	32

COC05: (new COCOMO data up to 2005):

rank	treatment	median	IQR
1	CART	42	61
1	COCOMO-II	46	146
1	Knear(1)	55	70
1	Knear(3)	63	99
1	COCONUT	66	34

NASA93: (NASA data up to 1993):

rank	treatment	median	IQR
1	COCONUT	36	38
1	COCOMO-II	38	39
2	CART	40	55
3	Knear(3)	54	66
3	Knear(1)	56	77

COC81: (original data from the 1981 COCOMO book):

rank	treatment	median	IQR
1	COCOMO-II	33	35
1	COCONUT	36	42
2	CART	66	95
3	Knear(3)	85	260
3	Knear(1)	87	236

Fig. 8. COCOMO vs standard methods. Displayed as per Figure 7.

The key observation in Figure 7 is that it is *not* the case that just using lines of code does better than parametric estimation. Also, when LOC(n) goes wrong, it goes very wrong indeed (as seen in the COC81 results, over double the median MRE error generated by COCOMO-II).

Another observation from Figure 7 is that, measured in terms of median MRE, COCONUT's local calibration is not better than untuned COCOMO. In only one data set (NASA93) did COCONUT have a lower median MRE than COCOMO-II but even in that case, Scott-Knott declared there was no significant difference between the COCOMO-II and COCONUT results.

On the other hand, sometimes the local calibration results exhibited far less variance than those of COCOMO-II. For example, in Figure 7's COC05 results, the IQR ranges for COCOMO-II and COCONUT were 146 and 34 respectively. This result (that local calibration reduces variance) repeats enough times in the subsequent experiments to make us recommend local calibration as a method for taming high variance in effort estimation.

B. COCOMO vs Other Methods

This section explores **RQ2: has parametric estimation been superseded by more recent estimation methods?** Also explored is **R3: Are the old parametric tunings irrelevant to more recent projects?**

Figure 8 shows a comparison of more standard effort estimation methods. (this figure is in the same format as Figure 7). Many of the results of Figure 8 repeat observations seen previously. For example, nothing was ever ranked better than COCOMO-II (sometimes CART or COCONUT had a slightly lower median MRE but that difference was small: $\leq 4\%$). From this result, we recommend that effort estimation researchers take care to benchmark their new method against older ones.

As to COCONUT, this method was usually ranked equaled to COCOMO-II. In once case (NASA10), COCOMO-II and COCONUT were ranked first and second but the median difference in their scores is very small (42 vs 47). Also, many other methods often had much larger variances. Hence, we can recommend some form of local calibration as a variance reduction tool.

From this data, we conclude that it is not always true the parametric estimation has been superseded by more recent innovations such as CART and Knear(n). Also, the COCOMO-II tunings from 2000 are useful not just for the projects used to make those tunings (all of COC81, plus some of NASA93) but also for projects completed up to a decade after those tunings (in NASA10).

C. COCOMO vs Simpler COCOMO

This section explores **RQ4: is parametric estimation expensive to deploy at some new site?**. To that end, we assess the impact a certain simplifications imposed onto COCOMO-II.

1) Range Reductions: One cost with deploying COCOMO in a new organization is the training effort required to generate consistent project rankings from

different analysts. If we could reduce the current six point scoring scale (very low, low, nominal, high, very high and extremely high) then there would be less scope for disagree about projects. Accordingly, for this experiment, we reduced the six point scale to just three:

- *Nominal*: same as before;
- *Above*: anything above nominal;
- *Below*: anything below nominal.

To do this, the tunings table of Figure 2 was altered. For each row, all values below nominal were replaced with their mean (and similarly with above-nominal values). For example, here are the tunings for *time* before and after being reduced to *below, nominal, above*:

range	vlow	low	*nominal*	high	vhigh	xhigh
before	1.22	1.09	1.00	0.93	0.86	0.80
reduced	1.15	1.15	1.00	0.863	0.863	0.863
	below			*above*		

2) Row Reductions: New COCOMO models are tuned only after collecting 100s of new examples. If that was not necessary, we could look forward to multiple COCOMO models, each tuned to different specialized (and small) samples of projects. Accordingly, we explore tuning COCOMO on very small data sets.

To implement this row reduction, training data was shuffled at random and training was conducted on all rows or on just the first four or eight rows rows (denoted *r4,r8* respectively). Note that, given the positive results obtained with *r8* we did not explore larger training sets.

3) Column Reduction: Prior results tell us that row reduction should be accompanied by column reduction. A study by Chen et al. [9] combines column reduction (that discards noisy or correlated attributes) with row reduction. Their results are very clear: as the number of rows shrink, *better* estimates come from using *fewer* columns. Miller [46] explains why this is so: the variance of a linear model learned by minimizing least-squares error decreases as the number of columns in the model decreases. That is, as the number of columns decrease, prediction reliability can increase (caveat: if you remove too much, there is no information left for predictions).

Accordingly, this experiment sorts the attributes in the training set according to how well they select for specific effort values. Let $x \in a_i$ denote the list of unique values seen for attribute a_i. Further, let there be N rows in the training data; let $r(x)$ denote the n rows containing x; and let $v(r(x))$ be the variance of the effort value in those rows. The values of "good" attributes select most for specific efforts; i.e. those attributes minimize:

$$E(\sigma, a_i) = \sum_{x \in a_i} \frac{n}{N} v(r(x)) \quad (3)$$

This experiment sorted all training data attributes by $E(\sigma, a_i)$ then kept the data in the *lower quarter* or *half* or *all* of the columns (denoted *c0.25* or *c0.5* or *c1* respectively). Note that, due to the results of Figure 7, LOC was excluded from column reduction.

NASA10 (new NASA data up to 2010):

rank	treatment	median	IQR
1	COCOMO-II	42	35
1	COCONUT:c0.5,r8	43	35
2	COCONUT	46	34
3	COCONUT:c1,r4	48	41
3	COCONUT:c1,r8	50	33
3	COCONUT:c0.25,r8	51	35
3	COCONUT:c0.5,r4	53	38
3	COCONUT:c0.25,r4	57	41

COC05 (new COCOMO data up to 2005):

rank	treatment	median	IQR
1	COCOMO-II	46	146
1	COCONUT:c0.5,r8	51	58
1	COCONUT:c0.25,r8	61	56
1	COCONUT:c0.5,r4	61	58
1	COCONUT	68	34
1	COCONUT:c1,r4	64	60
1	COCONUT:c1,r8	74	45
1	COCONUT:c0.25,r4	80	58

NASA93 (NASA data up to 1993):

rank	treatment	median	IQR
1	COCONUT	36	38
1	COCOMO-II	38	39
1	COCONUT:c0.5,r8	44	53
1	COCONUT:c0.5,r4	49	61
1	COCONUT:c0.25,r4	52	67
2	COCONUT:c1,r8	52	61
2	COCONUT:c1,r4	54	70
2	COCONUT:c0.25,r8	55	52

COC81 (original data from the 1981 COCOMO book):

rank	treatment	median	IQR
1	COCOMO-II	33	35
1	COCONUT	37	42
2	COCONUT:c1,r8	45	44
3	COCONUT:c0.5,r8	59	43
3	COCONUT:c0.25,r8	61	51
4	COCONUT:c1,r4	76	60
4	COCONUT:c0.5,r4	78	30
4	COCONUT:c0.25,r4	82	65

Fig. 9. COCOMO vs simpler COCOMO. MRE values. Displayed as per Figure 7.

4) Results: Figure 9 compares results found when use either *all* or some *reduce* set of ranges, rows, and columns. Note our nomenclature: the COCONUT:c0.5,r8 results are those seen after training on eight randomly selected training examples reduced to *below, nominal, above*, while ignoring 50% of the columns.

In Figure 9, all the *r4* results are ranked comparatively worse than the other treatments. That is, thse results do not condone learning from just four projects.

On the other hand Figure 9 suggests that it is defensible to learn a COCOMO model from eight projects. All the *r8* results are top-ranked with the exception of the COC81 results (but even there, the absolutely difference between the top *r8* results are standard COCOMO is very small).

Overall, Figure 9 suggests that the modeling effort associated with COCOMO-II could be reduced. Hence, it need not be expensive to deploy parametric esti-

mation at some new site. Projects attributes do not need to be specified in great detail: a simple three point scale will suffice: *below, nominal, above*. As to how much data is required for modeling, the results from COCONUT:c0.5,r8 are ranked either the same as COCOMO-II or (in the case of COC81) fall very close to the median and IQR seen for COCOMO-II. That is, a mere eight projects can suffice for calibration. Hence, it should be possible to quickly build many COCOMO-like models for various specialized sub-groups using just a three-point scale

That said, some column pruning should be employed when working with very small training sets (e.g. the eight rows used in Figure 9. Note that in all data sets that generated multiple rankings, the *c1* results (that used all the attributes) were not top-ranked. That is, in a result that might have been predicted by Miller or Chen et al., when working with just a few rows it is useful to reflect on what columns might be ignored.

D. COCOMO with Incorrect Size Estimates

This section explores **RQ5: Are parametric estimates unduly sensitive to errors in the size estimate?**

Before endorsing an KLOC-based estimation method, it is important to understand the effects of noise within the KLOC samples. Test projects to be estimated may have noisy KLOC values if the development team incorrectly guesstimated the size of the code. Training data may have noisy KLOC for many reasons such as

- How was reused code accounted for in the KLOC?
- Or were LOC counts based on end-statement or end-of-line symbols?
- Or how were lines of comments handled?

Another factor that introduces noise into training and test data are systems built from multiple languages. To make estimates from those kind of systems, KLOC in one language needs to be translated (in a possibly incorrect way) to KLOC in another language.

In theory, the problem of noisy KLOC measures seem particularly acute in our work. The core of COCOMO is an estimate that is exponential on KLOC. This means that KLOC will be magnified in a non-linear way). Also, if we train on just eight rows, as proposed above, then any noise in that small training data could be highly detrimental to the estimation process.

On the other hand, the coefficients on the exponential term in COCOMO equation are not large: $e = b + 0.01 * \sum_i SF_i$ where the default value for b is less than one (0.91) and the nominal values for SF_i sum to less than 20. In that default case $e = 0.91 + 0.01 * 20 = 0.912$ so KLOC errors may not be unduly magnified. However, for other non-default values, the coefficient is larger ($e_{max} = 1.22$) so it is important to check the effects of noise using real-world data.

NASA10 (new NASA data up to 2010): c

rank	treatment	median	IQR
1	COCOMO-II	42	35
1	COCONUT:c*0.5r=8n/4	46	36
1	COCOMO-IIn/2	50	47
2	COCONUT:c*0.5,r=8	54	34
2	COCONUT:c*0.5,r=8n/2	55	45
2	COCOMO-IIn/4	58	45

COC05 (new COCOMO data up to 2005):

rank	treatment	median	IQR
1	COCOMO-II	46	146
1	COCONUT:c*0.5,r=8n/2	60	53
1	COCOMO-IIn/2	63	203
1	COCONUT:c*0.5r=8n/4	65	42
1	COCONUT:c*0.5,r=8	77	51
2	COCOMO-IIn/4	110	280

NASA93 (NASA data up to 1993):

rank	treatment	median	IQR
1	COCOMO-II	38	39
1	COCOMO-IIn/2	41	43
1	COCOMO-IIn/4	47	45
1	COCONUT:c*0.5,r=8n/2	54	50
1	COCONUT:c*0.5,r=8	57	57
1	COCONUT:c*0.5r=8n/4	58	45

COC81 (original data from the 1981 COCOMO book):

rank	treatment	median	IQR
1	COCOMO-II	33	35
2	COCOMO-IIn/2	47	46
2	COCOMO-IIn/4	48	44
2	COCONUT:c*0.5,r=8n/2	53	45
2	COCONUT:c*0.5r=8n/4	57	44
2	COCONUT:c*0.5,r=8	65	51

Fig. 10. LOC noise results. MRE values. Displayed as per Figure 7.

To check the effects of noise, we repeated the reduction experiments of the last section while also injecting noise into the KLOC values. That is, as above, (1) the ranges were reduced to three; (2) half the columns were reduced; (3) we trained on only eight randomly selected projects; and (4) prior to train and test, all KLOC values were adjusted to

$$KLOC = KLOC * ((1 - n) + (2 * n * r))$$

where $n \in \{0.25, 0.5\}$ is the level of noise we are exploring and r is a random number $0 \leq r \leq 1$.

The results are shown in Figure 10. Any result marked with *n/2* or *n/4* shows what happens when the KLOCs were varied by 50% or 25% respectively. In only one case (COC81) were the noisy results statistically different from using data without noise. That is, the parametric estimation method being recommended here is not unduly effected by noise where the KLOC values vary up to 50% of their original value.

V. Threats to Validity

The above results were based with certain settings for some experiments on some data. For example, in the previous section, we used noise at levels 25% and 50%.

Clearly, these results may not hold if a wider range of settings for (e.g.) noise are explored.

Another source of bias in this study are the learners used for the defect prediction studies. Data mining is a large and active field and any single study can only use a small subset of the known data mining algorithms. The case for the learners used in this study was made in §III-C.

Questions of validity also arise in terms of how the projects (data-sets) are chosen for our experiments. While we used all the data sets that could be shared between our team, it is not clear if our results would generalize to other as yet unstudied data-sets. One the other hand, in terms of the parametric estimation literature, this is one of the most extensive and elaborate studies yet published.

On the matter of our data- there is a clear bias in our sample: all these projects are described in terms of the COCOMO attributes. Clearly, when more data becomes available, collected by different teams, we plan to repeat this study.

That said, it is not clear that the data sets used in this study were somehow simplistic or unchallenging. We saw that (e.g.) the COCOMO-II exhibited very large variances when applied to the COCO05 data. Further, supposedly state-of-the art methods had difficulty with generating good predictions with some of these data sets (recall the poor CART result for COC81 and NASA10). We therefore believe that these data sets are complex enough to serve as a stress test for different effort estimation methods.

VI. Conclusion

The past few decades have seen a long line of innovative methods applied to effort estimation. This paper has compared a sample of those methods to a decades-old parametric estimation method.

We found that:

- **RQ1**: parametric estimation is indeed better just using LOC (see §IV-A);
- **RQ2**: new innovations in effort estimation have not superseded parametric estimation (see §IV-B);
- **RQ3**: Old parametric tunings are not out-da¡ted (see §IV-B);
- **RQ4**: It is possible to simplify parametric estimation with some range, row and column pruning to reduce the cost of deploying those methods at a new site (see §IV-C);
- **RQ5**: Parametric estimation methods like COCOMO that assume effort is exponential on lines of code are *not* unduly sensitive to errors in the LOC measure (see §IV-D);.

Hence, we conclude that in 2015 is still valid and recommended practice to *first* try parametric estimation, perhaps augmented with a local calibration method like COCONUT and Equation 3's column pruner.

Having made that case, we need to add that in practice, approaches like expert judgment [19] and playing-poker for agile estimation [49] have their home ground, but that is not enough of a reason to reject decades of research into parametric estimation and COCOMO. Best practices and empirical studies show that different methods can work together to reduce biases [36]. What we hope we have shown here is that it would be useful for those best practices to *include*, and not *replace*, parametric estimation.

Acknowledgment

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

References

[1] David W. Aha, Dennis Kibler, and Marc K. Albert. Instance-based learning algorithms. *Mach. Learn.*, 6(1):37–66, January 1991.

[2] A. Arcuri and L. Briand. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *ICSE'11*, pages 1–10, 2011.

[3] Martin Auer, Adam Trendowicz, Bernhard Graser, Ernst Haunschmid, and Stefan Biffl. Optimal project feature weights in analogy-based cost estimation: Improvement and limitations. *IEEE Trans. Softw. Eng.*, 32:83–92, 2006.

[4] Dan Baker. A hybrid approach to expert and model-based effort estimation. Master's thesis, Lane Department of Computer Science and Electrical Engineering, West Virginia University, 2007. Available from https://eidr.wvu.edu/etd/documentdata.eTD?documentid=5443.

[5] R. Black, R. Curnow, R. Katz, and M. Bray. Bcs software production data, final technical report radc-tr-77-116. Technical report, Boeing Computer Services, Inc., March 1977.

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

[7] 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.

[8] C.J. Burgess and Martin Lefley. Can genetic programming improve software effort estimation? a comparative evaluation. *Information and Software Technology*, 43(14):863–873, December 2001.

[9] Zhihao Chen, Barry Boehm, Tim Menzies, and Daniel Port. Finding the right data for software cost modeling. *IEEE Software*, 22:38–46, 2005.

[10] Zhihoa Chen, Tim Menzies, and Dan Port. Feature subset selection can improve software cost estimation. In *PROMISE'05*, 2005. Available from http://menzies.us/pdf/05/fsscocomo.pdf.

[11] A. Corazza, S. Di Martino, F. Ferrucci, C. Gravino, F. Sarro, and E. Mendes. How effective is tabu search to configure support vector regression for effort estimation? In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, PROMISE '10, pages 4:1–4:10, 2010.

[12] R. Cordero, M. Costamagna, and E. Paschetta. A genetic algorithm approach for the calibration of cocomo-like models. In *12th COCOMO Forum*, 1997.

[13] J. B. Dabney. Return on investment for IV&V, 2002-2004. NASA funded study. Results Available from http://sarpresults.ivv.nasa.gov/ViewResearch/24.jsp.

[14] Karel Dejaeger, Wouter Verbeke, David Martens, and Bart Baesens. Data mining techniques for software effort estimation: A comparative study. *IEEE Transactions on Software Engineering*, 38:375–397, 2012.

[15] Bradley Efron and Robert J Tibshirani. *An introduction to the bootstrap*. Mono. Stat. Appl. Probab. Chapman and Hall, London, 1993.

[16] F. Freiman and R. Park. Price software model - version 3: An overview. In *Proceedings, IEEE-PINY Workshop on Quantitative Software Models, IEEE Catalog Number TH 0067-9*, pages 32–41, October 1979.

[17] J. Herd, J. Postak, W. Russell, and J. Stewart. Software cost estimation study-study results, final technical report, radc-tr-77-220. Technical report, Doty Associates, June 1977.

[18] R. Jensen. An improved macrolevel software development resource estimation model. In *5th ISPA Conference*, pages 88–92, April 1983.

[19] M. Jørgensen and T.M. Gruschke. The impact of lessons-learned sessions on effort estimation and uncertainty assessments. *Software Engineering, IEEE Transactions on*, 35(3):368 –383, May-June 2009.

[20] M. Jørgensen and M. Shepperd. A systematic review of software development cost estimation studies, January 2007. Available from http://www.simula.no/departments/engineering/publications/J{\o}rgensen.2005.12.

[21] G. Kadoda, M. Cartwright, L. Chen, and M. Shepperd. Experiences using casebased reasoning to predict software project effort, 2000.

[22] Vigdis By Kampenes, Tore Dybå, Jo Erskine Hannay, and Dag I. K. Sjøberg. A systematic review of effect size in software engineering experiments. *Information & Software Technology*, 49(11-12):1073–1086, 2007.

[23] Jacky Wai Keung. Empirical evaluation of analogy-x for software cost estimation. In *ESEM '08: International Symposium on Empirical Software Engineering and Measurement*, pages 294–296, New York, NY, USA, 2008. ACM.

[24] Jacky Wai Keung and Barbara Kitchenham. Experiments with analogy-x for software cost estimation. In *ASWEC '08: Proceedings of the 19th Australian Conference on Software Engineering*, pages 229–238, Washington, DC, USA, 2008. IEEE Computer Society.

[25] Jacky Wai Keung, Barbara A. Kitchenham, and David Ross Jeffery. Analogy-x: Providing statistical inference to analogy-based software cost estimation. *IEEE Trans. Softw. Eng.*, 34(4):471–484, 2008.

[26] C. Kirsopp and M. Shepperd. Making inferences with small numbers of training sets. *IEEE Proc.*, 149, 2002.

[27] E. Kocaguneli and T. Menzies. Software effort models should be assessed via leave-one-out validation. *Journal of Systems and Software*, 2013, to appear.

[28] E. Kocaguneli, T. Menzies, A. Bener, and J. Keung. Exploiting the essential assumptions of analogy-based effort estimation. *IEEE Transactions on Software Engineering*, 28:425–438, 2012. Available from http://menzies.us/pdf/11teak.pdf.

[29] E. Kocaguneli, Tim Menzies, and J. Keung. On the value of ensemble effort estimation. *IEEE Transactions on Software Engineering*, 2012. Available from http://menzies.us/pdf/11comba.pdf.

[30] Ekrem Kocaguneli, Tim Menzies, Jacky Keung, David Cok, and Ray Madachy. Active learning and effort estimation: Finding the essential content of software effort estimation data. *IEEE Transactions on Software Engineering*, 39(8):1040–1053, 2013.

[31] Ekrem Kocaguneli, Tim Menzies, and Emilia Mendes. Transfer learning in effort estimation. *Empirical Software Engineering*, pages 1–31, 2014.

[32] Ekrem Kocaguneli, Thomas Zimmermann, Christian Bird, Nachiappan Nagappan, and Tim Menzies. Distributed development considered harmful? In *ICSE*, pages 882–890, 2013.

[33] Jingzhou Li and Guenther Ruhe. A comparative study of attribute weighting heuristics for effort estimation by analogy. *International Symposium on Empirical Software Engineering*, page 74, 2006.

[34] Jingzhou Li and Guenther Ruhe. Decision support analysis for software effort estimation by analogy. In *PROMISE '07: Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, page 6, 2007.

[35] Jingzhou Li and Guenther Ruhe. Analysis of attribute weighting heuristics for analogy-based software effort estimation method aqua+. *Empirical Softw. Engg.*, 13:63–96, February 2008.

[36] Qi Li, Qing Wang, Ye Yang, and Mingshu Li. Reducing biases in individual software effort estimations: a combining approach. In *Proceedings of the Second International Symposium on Empirical Software Engineering and Measurement, ESEM 2008, October 9-10, 2008, Kaiserslautern, Germany*, pages 223–232, 2008.

[37] Y Li, M Xie, and T Goh. A study of project selection and feature weighting for analogy based software cost estimation. *Journal of Systems and Software*, 82:241–252, 2009.

[38] Y. Li, M. Xie, and Goh T. A study of the non-linear adjustment for analogy based software cost estimation. *Empirical Software Engineering*, pages 603–643, 2009.

[39] C. Lokan and E. Mendes. Cross-company and single-company effort models using the isbsg database: a further replicated study. In *The ACM-IEEE International Symposium on Empirical Software Engineering, November 21-22, Rio de Janeiro*, 2006.

[40] C. Lokan and E. Mendes. Applying moving windows to software effort estimation. In *Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on*, pages 111–122, 2009.

[41] Emilia Mendes, Ian D. Watson, Chris Triggs, Nile Mosley, and Steve Counsell. A comparative study of cost estimation models for web hypermedia applications. *Empirical Software Engineering*, 8(2):163–196, 2003.

[42] Tim Menzies, Andrew Butcher, David R. Cok, Andrian Marcus, Lucas Layman, Forrest Shull, Burak Turhan, and Thomas Zimmermann. Local versus global lessons for defect prediction and effort estimation. *IEEE Trans. Software Eng.*, 39(6):822–834, 2013. Available from http://menzies.us/pdf/12localb.pdf.

[43] Tim Menzies, Zhihao Chen, Jairus Hihn, and Karen Lum. Selecting best practices for effort estimation. *IEEE Transactions on Software Engineering*, November 2006. Available from http://menzies.us/pdf/06coseekmo.pdf.

[44] Tim Menzies, D. Port, Z. Chen, J. Hihn, and S. Stukes. Validation methods for calibrating software effort models. In *Proceedings, ICSE*, 2005. Available from http://menzies.us/pdf/04coconut.pdf.

[45] Tim Menzies and Martin Shepperd. Special issue on repeatable results in software engineering prediction. *Empirical Software Engineering*, 17(1-2):1–17, 2012.

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

[47] Leandro L. Minku and Xin Yao. How to make best use of cross-company data in software effort estimation? In *ICSE'14*, pages 446–456, 2014.

[48] Nikolaos Mittas and Lefteris Angelis. Ranking and clustering software cost estimation models through a multiple comparisons algorithm. *IEEE Trans. Software Eng.*, 39(4):537–551, 2013.

[49] Kjetil Molokken-Pstvold, Nils Christian Haugen, and Hans Christian Benestad. Using planning poker for combining expert estimates in software projects. *Journal of Systems and Software*, 81:21062117, December 2008.

[50] Ingunn Myrtveit, Erik Stensrud, and Martin Shepperd. Reliability and validity in comparative studies of software prediction models. *IEEE Trans. Softw. Eng.*, 31(5):380–391, May 2005.

[51] Ingunn Myrtveit, Erik Stensrud, and Martin Shepperd. Reliability and validity in comparative studies of software prediction models. *IEEE Transactions on Software Engineerining*, 31(5):380–391, May 2005.

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

[53] Carol Passos, Ana Paula Braun, Daniela S. Cruzes, and Manoel Mendonca. Analyzing the impact of beliefs in software project practices. In *ESEM'11*, 2011.

[54] D. Posnett, V. Filkov, and P. Devanbu. Ecological inference in empirical software engineering. In *Proceedings of ASE'11*, 2011.

[55] L. Putnam. A macro-estimating methodology for software development. In *Proceedings, IEEE COMPCON76 Fall*, pages 38–43, September 1976.

[56] Mary Shaw. The coming-of-age of software architecture research. In *Proceedings of the 23rd International Conference on Software Engineering*, ICSE '01, pages 656–, Washington, DC, USA, 2001. IEEE Computer Society.

[57] 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.

[58] Martin J. Shepperd and Steven G. MacDonell. Evaluating prediction systems in software project estimation. *Information & Software Technology*, 54(8):820–827, 2012.

[59] Spareref.com. Nasa to shut down checkout & launch control system, August 26, 2002. http://www.spaceref.com/news/viewnews.html?id=475.

[60] Fiona Walkerden and Ross Jeffery. An empirical study of analogy-based software effort estimation. *Empirical Softw. Engg.*, 4(2):135–158, 1999.

[61] C. Walston and C. Felix. A method of programming measurement and estimation. *IBM Systems Journal*, (1):54–77, 1977.

[62] R. Wolverton. The cost of developing large-scale software. *IEEE Trans. Computers*, pages 615–636, June 1974.