Noname manuscript No.
(will be inserted by the editor)

Negative Results for Software Effort Estimation

Tim Menzies, Ye Yang, George Mathew, Barry Boehm, Jairus Hihn

Abstract *Context:* More than half the literature on software effort estimation (SEE) focuses on comparisons of new estimation methods. Surprisingly, there are no studies comparing state of the art latest methods with decades-old approaches.

Objective: To check if new SEE methods generated better estimates than older methods.

Method: Firstly, collect effort estimation methods ranging from "classical" COCOMO (parametric estimation over a pre-determined set of attributes) to "modern" (reasoning via analogy using spectral-based clustering plus instance and feature selection). *Secondly*, catalog the list of objections that lead to the development of post-COCOMO estimation methods. *Thirdly*, characterize each of those objections as a comparison between newer and older estimation methods. *Fourthly*, using four COCOMO-style data sets (from 1991, 2000, 2005, 2010), run those comparisons experiments. *Fifthly*, compare the performance of the different estimators using a Scott-Knott procedure using (i) the A12 effect size to rule out "small" differences and (ii) a 99% confident bootstrap procedure to check for statistically different groupings of treatments). *Sixthly*, repeat the above for some non-COCOMO data sets.

Results: For the non-COCOMO data sets, our newer estimation methods performed better than older methods. However, the **major negative result** of this paper is that for the COCOMO data sets, nothing we used did any better than Boehm's original procedure.

Conclusions: In some projects, it is not possible to collect effort data in the COCOMO format recommended by Boehm. For those projects, we recommend using newer effort estimation methods. However, when COCOMO-style attributes are available, we strongly recommend using that data since the experiments of this paper show that, at least for effort estimation, *how data is collected* is more important than *what learner is applied to that data*.

Categories/Subject Descriptors: D.2.9 [Software Engineering]: Time Estimation; K.6.3 [Software Management]: Software Process

Keywords: effort estimation, COCOMO, CART, nearest neighbor, clustering, feature selection, prototype generation, bootstrap sampling, effect size, A12.

T. Menzies, G. Mathew
CS, North Carolina State Univ., USA E-mail: tim.menzies@gmail.com, E-mail: george.meg91@gmail.com

Y. Yang
SSE, Stevens Inst., USA E-mail: yangye@gmail.com

B. Boehm
CS, Univ. of Southern California, USA E-mail: barryboehm@gmail.com

J. Hihn
JPL, CalTech, USA E-mail: jairus.hihn@jpl.nasa.gov

1 Introduction

This paper is about a negative result in software effort estimation– specifically, even after decades of research in this area, we are unable to do better than a parametric estimation method proposed in 2000 [6].

For pragmatic and methodological reasons,it is important to report such negative results. Pragmatically, it is important for industrial practitioners to know that (sometimes) they do not need to waste time straining to understand bleeding-edge technical papers. In the following, we precisely define the class of project data that *does not* respond well to bleeding-edge effort estimation techniques. For those kinds of data sets, practitioners can be rest assured that it is reasonable and responsible and useful to use simple traditional methods.

Also, methodologically, it is important to acknowledge mistakes. According to Karl Popper (a prominent figure in the philosophy of science [59]), the "best" theories are the ones that have best survived vigorous debate. Having been engaged in some high-profile debates (in the field of software analytics [46]), we assert that such criticisms are very useful since they help a researcher (1) find flaws in old ideas and (2) evolve better new ideas. That is, finding and acknowledging mistakes should be regarded as a routine part of standard operations procedure for science.

Given the above, it is troubling that there are very few failure reports in the field of software analytics. Some authors include a "Erratta" section in the papers that patch older conclusions. However, such reports are infrequent (evidence: we know of only two such report in the last five years of conferences on software analytics [47, 54]). Given the complexity of software analytics, this absence of such failure reports is highly suspicious.

Why are these reports so rare? There are many possible reasons and here we speculate on two possibilities. Firstly, such negative reports may not be acknowledged as "worthwhile" by the community. Forums such as this special issue are very rare (which is why this issue is so important). Secondly, it is not standard practice in software analytics for researchers to benchmark their latest results against some supposedly simpler "straw man" method. In his textbook on *Empirical Methods in AI*, Cohen [13] strongly advises such "straw man" comparisons, since sometimes, they reveal that the supposedly superior method is actually overly complex. Hence we take care to compare methods against simpler alternative.

The rest of this paper discusses our negative result, and its scope. We will show that there exists some effort estimation data sets for which it is useful to use the latest generation of effort estimation techniques. However, we will also show that there exists a second class of data set for which very old methods do just as well as anything else.

That second class contains data expressed in terms of the COCOMO ontology: 22 attributes describing a software project, as well as aspects of its personnel, platform and product features[1]. We will show that (given this diverse sample of data types collected from a project) Boehm's 2000 model works as well (or better) than everything else we tried. Hence, we strongly recommend that if that kind of data is available, then it should be collected and it should be processed using Boehm's 2000 COCOMO model.

To guide our exploration, this paper asks five research questions. These questions have been selected based on our experience debating the merits of COCOMO vs alternate methods. Based on our experience, we assert that each of the following questions has been used to motivate the development of some alternate to the standard COCOMO-II model:

RQ1: Is parametric estimation no better than using just Lines of Code measures? (an often heard, but rarely tested, comment).

[1] For full details on these attributes, see §4 of this paper.

RQ2: Has parametric estimation been superseded by more recent estimation methods? We apply our "best" learner, as well as 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 to a wide range of projects dating 1970 to 2010.

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

RQ5: Are parametric estimates unduly sensitive to errors in the size estimate? In the context of RQ4, we check what happens if there are large errors in the "thousand lines of code"(KLOC) estimate.

To explore these questions, we use COCOMO since its internal details have been fully published [7]. Also, we can access a full implementation of the 1998 COCOMO model. Further, we have access to numerous interesting COCOMO data sets: see Figure 1 and Figure 2. With one exception, our learning experiments do not use the data that generated standard COCOMO. That exception is the COC81 data– which lets us compare new methods against the labor intensive methods used to make standard COCOMO– see Figure 2.

Using that data, the experiments of this paper conclude that the answer to all our research questions is "no". The RQ1 experiments show that good estimates use many variables and poorer estimates result from some trite calculation based on KLOC. Hence, we can make some degree of error in our KLOC estimates without damaging the overall estimation process (see RQ5).

Types of projects	COC81	NASA93	COC05	NASA10
Avionics		26	10	17
Banking			13	
Buss.apps/databases	7	4	31	
Control	9	18	13	
Human-machine interface	12			
Military, ground			8	
Misc	5	4	5	
Mission Planning		16		
SCI scientific application	16	21	11	
Support tools,	7			
Systems	7	3	2	

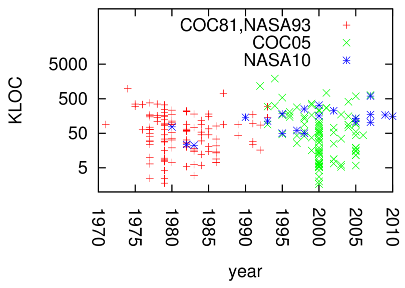


Fig. 1: Projects used by the learners in this study. Figure 3 shows project attributes. COC81 is the original data from 1981 COCOMO book [5]. This comes from projects dating 1970 to 1980. NASA93 is NASA data collected in the early 1990s about software that supported the planning activities for the International Space Station. Our two other data sets are COC05 and NASA10 (these data sets are proprietary and cannot be released to the research community). The non-proprietary data (COC81 and NASA93) is available at http://openscience.us/repo.

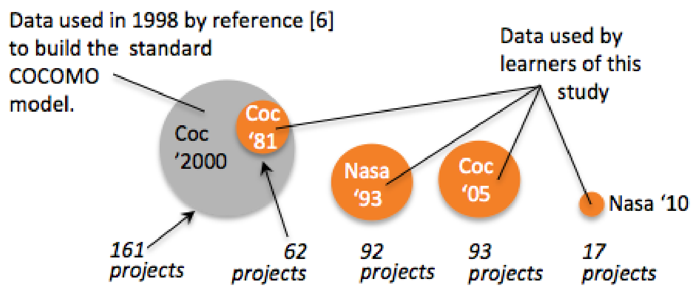

Data used in 1998 by reference [6] to build the standard COCOMO model.
Data used by learners of this study
Coc '2000
Coc '81
Nasa '93
Coc '05
Nasa '10
161 projects
62 projects
92 projects
93 projects
17 projects

Fig. 2: Projects in this study (COC81 is a subset of COCOMO-II).

As to the other research questions (R2,R3,R4), those results mean that the continued use of parametric estimation can still be endorsed– at least for data expressed in terms of the 22 COCOMO attributes. What COCOMO, or other parametric estimation models, can offer largely contains the fundamentals for making estimation decisions from expert-Delphi, as well as well calibrated tuning factors from over 40 years of industrial data. We would advise researchers *not* to ignore these fundamentals.

2 About Effort Estimation

2.1 History

Accurately estimating software development effort is of vital importance. Under-estimation can cause schedule and budget overruns as well as project cancellation [66]. Over-estimation delays funding to other promising ideas and organizational competitiveness [34]. Hence, there is a long history of researchers exploring software effort estimation; e.g. [4, 5, 9, 19,

	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. 3: COCOMO-II attributes.

20, 22, 24, 45, 57, 61, 64, 68–70]. In 2007, Jorgensen and Shepperd reported on hundreds of research papers dating back to the 1970s devoted to the topic, over half of which propose some innovation for developing new estimation models [24]. Since then, many more such papers have been published; e.g. [14, 29–31, 33, 35, 36, 39, 41, 42, 49, 51].

In the 1970s and 1980s, this kind of research was focused on *parametric estimation* as done by Putnam and others [4, 5, 19, 20, 69, 70]. For example, Boehm's COnstructive COst MOdel (COCOMO) model [5] assumes that effort varies exponentially on size as seen in this parametric form: $\mathit{effort} \propto a \times KLOC^b$. 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, with minor tweaks [48]. COCOMO is a parametric method; i.e. it is a *model-based* method that (a) assumes that the target model has a particular structure, then (b) uses model-based methods to fill in the details of that structure (e.g. to set some tuning parameters).

Since that work on parametric estimation, researchers have innovated other methods based on regression trees [64] case-based-reasoning [64], spectral clustering [44], genetic algorithms [9, 15], etc. These methods can be augmented with "meta-level" techniques like tabu search [14], feature selection [11], instance selection [33], feature synthesis [49], active learning [35], transfer learning [36]. temporal learning [43, 51], and many more besides.

2.2 Current Practice

In her keynote address to ICSE'01, Mary Shaw [62] 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. Given that, it would be reasonable to expect commercial adoption of the 1990s estimation work on regression trees [64] or case-based-reasoning [64]. However, this has not happened. 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 [16].
- In our work with the Chinese and the United States software industry, we saw an almost exclusive use of parametric estimation tools such as those offered by Price Systems (pricesystems.com) and Galorath (galorath.com).
- Professional societies, handbooks and certification programs are mostly developed around parametric estimation methods and tools; e.g. see the International Cost Estimation and Analysis Society; the NASA Cost Symposium; the International Forum on COCOMO and Systems/Software Cost Modeling (see the websites goo.gl/u3q9Nq, goo.gl/8jxPrb, goo.gl/O01Pc6).

2.3 But Does Anyone Use COCOMO?

Two of the myths of effort estimation is that (1) no one used model-based estimation like COCOMO; and (2) estimates are always better done using expert-based guess-timation, a.k.a. *Delphi-based* methods[2].

These myths are misleading. As seen above, model-based parametric methods are widely used in industry and are strongly advocated by professional societies. Also, while it is true that Delphi-based estimation is a common practice [6], this is not to say that this should be recommended as the *best* or *only* way to make estimates:

- Jorgensen [25] reviews studies comparing model- and Delphi- based estimation and concludes that there there is no clear case that Delphi-methods are better.
- Valerdi [67] lists the cognitive biases that can make an expert offer poor Delphi-estimates.

[2] The mythological oracle of Delphi spoke for the god Apollo to answer questions about colonization, religion, and power.

– Passos et al. show that many commercial software engineers generalize from their first few projects for all future projects [58].
– Jorgensen & Gruschke [23] document how commercial "gurus" rarely use lessons from past projects to improve their future Delphi-estimates. They offer examples where this failure to revise prior beliefs leads to poor Delphi-based estimates.

Much research has concluded that the best estimations come from *combining* the predictions from *multiple oracles* [3, 12, 34, 67].

Note that it is far easy to apply this double-check strategy using Delphi+model-based methods than by comparing the estimates from multiple Delphi teams. For example, all the model-based methods studied in this paper can generate estimates in just a few seconds. In comparison, Delphi-based estimation is orders of magnitude slower– as seen in Valerdi's COSYSMO Delphi-method. While a strong proponent of this approach, Valerdi concedes that "(it is) extremely time consuming when large sample sizes are needed" [67]. For example, he once recruited 40 experts to three Delphi sessions, each of which ran for three hours. Assuming a 7.5 hour day, then that study took $3 * 3 * 40/7.5 = 48\ days$.

COSYSMO is an elaborate Delphi-based method. An alternate, more lightweight Delphi-method is "planning poker" [53] where participants offer anonymous "bids" on the completion time for a project. If the bids are widely divergent, then the factors leading to that disagreement elaborated and debated. This cycle of bid+discuss continues until a consensus has been reached.

While planning poker is widely advocated in the agile community, there are surprisingly few studies assessing this method (one rare exception is [53]). Further, planning poker is used to assess effort for particular tasks in the scrum backlog– which is a different and simpler task than the *initial* estimation of large-scale projects. This is an important issue since, for larger projects, the initial budget allocation may require a significant amount of intra-organizational lobbying between groups with competing concerns. For such large-estimate-projects, it can be challenging to change the initial budget allocation. Hence, it is important to get the initial estimate as accurate as possible.

2.4 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 TRW's software business [70], but it did not relate well to the full range of embedded software, command and control software, and engineering and scientific software involved in TRW's 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 find the relative significance of various cost drivers. Combining local expertise and data, plus some prior results such as [4, 20, 61, 69], and early versions of the RCA PRICE S model [19], a model called SCEP was created (Software Cost Estimation Program). Except for one explainable outlier, the estimates for 20 projects with solid data were within 30% of the actuals, 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

```
_  = 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
  sfs ems, kloc  = 0,1,22
  scaleFactors, effortMultipliers = 5, 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. 4: 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 3.

courses on software cost estimation, Boehm was able to gather 63 data points that could be 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 [5]. 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). Using that new data, Boehm and his colleagues developed the *tunings* shown in Figure 4 that map the project descriptors (very low, low, etc) into the specific values used in the COCOMO-II model (released in 2000 [7]):

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

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

Note that, from Equation 1, the minimum effort is bounded by the *sum* of the minimum scale factors and the *product* of the minimum effort multipliers. Similar expressions hold for the maximum effort estimate. Hence, for a given KLOC, the range of values is given by:

$$0.18 * KLOC^{0.97} \leq \mathit{effort} \leq 154 * KLOC^{1.23}$$

Dividing the minimum and maximum values results in an expression showing how effort can vary for any given KLOC.:

$$154/0.18 * KLOC^{1.23-0.97} = 856 * KLOC^{0.25} \tag{2}$$

2.5 COCOMO and Local Calibration

When local data is scarce, approximations can be used to tune a model using just a handful of examples. COCOMO *local calibration* procedure, adjusts the impact of the scale factors and effort multipliers by tuning the a, b values of Equation 1 while keeping the other values of the tuning matrix constant as shown in Figure 4. Effectively, local calibration trims a 23 variable model into a model with two variables: (one to adjust the linear effort multipliers, and another to adjust the exponential scale factors).

Menzies' preferred local calibration procedure is the COCONUT procedure of Figure 5 (first written in 2002 and first published in 2005 [48]). 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.

3 Experimental Methods

In this section, we discuss the methods used to explore the reserach questions defined in the introduction.

3.1 Choice of Experimental Rig

"Ecological inference" is the conceit that what holds for all, also holds for parts of the population [44, 60]. To avoid ecological inference, our rig in Figure 6 runs separately for each data set.

Since some of our methods include a stochastic algorithm (the COCONUT algorithm of Figure 5), we repeat our experimental rig $N = 10$ times (10 was selected since, after experimentation, we found our results looked the same at $N = 8$ and $N = 16$).

It is important to note that Figure 6 is a "leave-one-out experiment"; i.e. training is conducted on all-but-one example, then tested on a "holdout" example not seen in training. This separation of training and testing data is of particular importance in this study. As shown in Figure 1, our data sets (NASA10, COC81, NASA93, and COC05) contain information on 17, 63, 92, and 93 projects, respectively. When fitted to the 24 parameters of the standard COCOMO model (shown in Figure 3), there may not be enough information to constrain the learning– which means that it is theoretically possible that data could be fitted to almost anything (including *spurious noise*). To detect such spurious models, it is vital to test the learned model against some outside source such as the holdout example.

```
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 = a1 - deltaA + 2 * deltaA * rand()
    bGuess = b1 - 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. 5: COCONUT tunes a, b of Figure 4's COCOMO function.

```
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. 6: The experimental rig used in this paper.

We assess performance via the magnitude of the relative error; i.e.

$$MRE = \frac{abs(actual - predicted)}{actual}. \tag{3}$$

Shepperd & MacDonnell [65] 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 & MacDonnell 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 in the next section.

3.2 Choice of Learners

Our LOC(n) "straw man" estimators just uses lines of code in the n nearest projects. For distance, we use:

$$dist(x,y) = \sqrt{\sum_i w_i(x_i - y_i)^2} \tag{4}$$

where x_i, y_i are values normalized 0..1 for the range min..max and w_i is a weighting factor (defaults to $w_i = 1$). When estimating for $n > 1$ neighbors, we combine estimates via the triangle function of Walkerden and Jeffery [68]; e.g.. for $loc(3)$, the estimate from the first, second and third closest neighbor with estimates a, b and c respectively is

$$\mathit{effort} = (50a + 33b + 17c)/100 \tag{5}$$

Apart from the LOC "straw man", we also compare COCOMO-II and COCONUT with CART, Knear(n), TEAK, and PEEKING2. These methods were selected, for the following reasons. TEAK and PEEKING2 represent recent innovations in effort estimation [33, 56]. CART and Knear(n) are more traditional methods that proved their value in the 1990s [64, 68]. That said, CART and Knear(n) still have currency: recent results from IEEE TSE 2008 and 2012 still endorse their use for effort estimation [17, 31, 34]). Also, according to the Shaw's timetable for industry adoption of research innovations (discussed in the introduction), CART and Knear(n) should now be mature enough for industrial use.

CART [8] 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 are averaged to generate the estimate.

Knear(n) estimates a new project's effort by a nearest neighbor method [64]. 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 5. Knear(n) is an example of CBR; i.e. *case-based reasoning*. CBR for effort estimation was first pioneered by Shepperd & Schofield in 1997 [64]. Since then, it has been used extensively in software effort estimation [2, 27, 29–32, 38–41, 64, 68]. There are several reasons for this. Firstly, it works even if the domain data is sparse [55]. Secondly, unlike other predictors, it makes no assumptions about data distributions or some underlying parametric model.

TEAK is built on the assumption that spurious noise leads to large variance in the recorded efforts [33]. TEAK's pre-processor removes such regions of high variance as follows. First, it applies greedy agglomerate clustering to generate a tree of clusters. Next, it reflects on the variance of the efforts seen in each sub-tree and discards the sub-trees with largest variance. Estimation is then performed on the surviving examples.

PEEKING2 [56] is a far more aggressive "data pruner" than TEAK and combines data reduction operators, feature weighting, and Principal Component Analysis(PCA). PEEKING2 is described in Figure 7.

3.3 Choice of Statistical Ranking Methods

The last line of our experimental rig shown in Figure 6 *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 [52]. 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

- PEEKING2's feature weighting scheme changes w_i in Equation 4 according to how much an attribute can divide and reduce the variance of the effort data (the *greater* the reduction, the *larger* the w_i score).
- PEEKING2's PCA tool uses an accelerated principle component analysis that synthesises new attributes $e_i, e_2, ...$ that extends across the dimension of greatest variance in the data with attributes d. PCA combines redundant variables into a smaller set of variables (so $e \ll d$) since those redundancies become (approximately) parallel lines in e space. For all such redundancies $i, j \in d$, we can ignore j since effects that change over j also change in the same way over i. PCA is also useful for skipping over noisy variables from *d*– these variables are effectively ignored since they do not contribute to the variance in the data.
- PEEKING2's prototype generator clusters the data along the dimensions found by accelerated PCA. Each cluster is then replaced with a "prototype" generated from the median value of all attributes in that cluster. Prototype generation is a useful tool for handling outliers: large groups of outliers get their own cluster; small sets of outliers get ignored via median prototype generation.
- PEEKING2 generates estimates for a test case by finding its nearest cluster, then the two nearest neighbors within that cluster (where "near" is computed using Equation 4 plus feature weighting). If these neighbors are found at distance $n_1, n_2, n_1 < n_2$ and have effort values E_1, E_2 then the final estimate is an extrapolation favoring the closest one:

$$n = n_i + n_2;\ estimate = E_1\frac{n_2}{n} + E_2\frac{n_1}{n}$$

Fig. 7: Inside PEEKING2 [56].

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

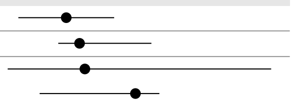
Note that Scott-Knott found little difference between rx5 and rx3. Hence, they have the same rank, even though their medians differ.

Scott-Knott is better than an all-pairs hypothesis test of all methods; e.g. six treatments can be compared $(6^2 - 6)/2 = 15$ ways. A 95% confidence test run for each comparison has a very low total confidence: $0.95^{15} = 46\%$. To avoid an all-pairs comparison, Scott-Knott only calls on hypothesis tests *after* it has found splits that maximize the performance 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 ($A12 \geq 0.6$).

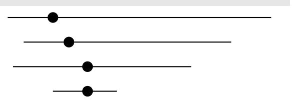
NASA10 (new NASA data up to 2010):

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



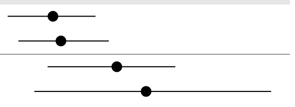
COC05 (new COCOMO data up to 2005):

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

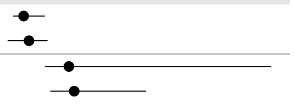


Fig. 8: COCOMO vs just lines of code. MRE values seen in leave-one-studies, repeated ten times. For each of the four tables in this figure, *better* methods appear *higher* in the tables. In these tables, median and IQR are the 50th and the (75-25)th percentiles. The IQR range is shown in the right column with black dot at the median. Horizontal lines divide the "ranks" found by our Scott-Knott+bootstrapping+effect size tests (shown in left column).

For a justification of the use of non-parametric bootstrapping, see Efron & Tibshirani [18, p220-223]. For a justification of the use of effect size tests see Shepperd & MacDonell [65]; Kampenes [28]; and Kocaguneli et al. [37]. 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. 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:

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

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 [1].

4 Results

4.1 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 8, this is not necessarily true. This figure is a comparative ranking for LOC(1) LOC(3), COCOMO-II and COCONUT. The rows of Figure 8 are sorted by the median MRE figures. 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).

The key feature of Figure 8 is that just using lines of code is *not* better than parametric estimation. Also, when LOC(n) goes wrong, it goes very wrong indeed (see the COC81 results: LOC(3) produces double the median MRE error generated by COCOMO-II).

Equation 2 explains why just using KLOC performs so badly. That equation had two components: KLOC raised to a small exponent (0.25), and a constant showing the influence of all other COCOMO variables. The large value of 856 for that second component indicates that many factors outside of KLOC influence effort. Hence, it is hardly surprising that just using KLOC is a poor way to do effort estimation.

Another observation from Figure 8 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 8'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.

4.2 COCOMO vs Other Methods

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

Figure 9 compares COCOMO and COCONUT with traditional effort estimation methods from the 1990s (CART and Knear(n)). In that comparison, nothing was ever ranked better than COCOMO-II (sometimes CART or COCONUT had a slightly lower median MRE but that difference was small: $\leq 4\%$).

Figure 10 compares COCOMO and COCONUT to more recent effort estimation methods (TEAK and PEEKING2). Once again, nothing was ever ranked better than COCOMO-II or COCONUT.

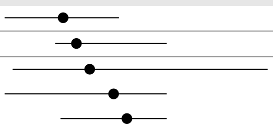
From these results, we recommend that effort estimation researchers take care to benchmark their new method against older ones.

As to COCONUT, this method was usually ranked equaled to COCOMO-II. In several cases COCOMO-II and COCONUT were ranked first and second but the median difference in their scores is very small: see NASA10 of Figure 9 and NASA93,COC81 of Figure 10 Also, many other methods often had much larger variances. Hence, we can recommend some form of local calibration as a variance reduction tool (e.g. compare COCONUT with COCOMO-II in COC05 of Figure 10).

From this data, we conclude that it is not always true the parametric estimation has been superseded by more recent innovations such as CART, Knear(n), TEAK or PEEKING2. Also, the COCOMO-II tunings from 2000 are useful not just for the projects prior to 200 (all of COC81, plus some of NASA93) but also for projects completed up to a decade after those tunings (NASA10).

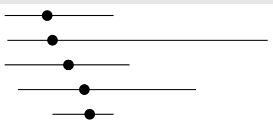
NASA10: (new NASA data up to 2010):

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



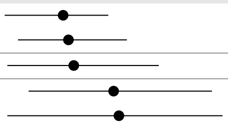
COC05: (new COCOMO data up to 2005):

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

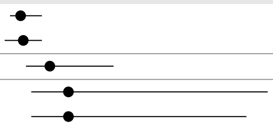


Fig. 9: COCOMO vs standard methods. Displayed as per Figure 8.

4.3 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.

4.3.1 Range Reductions

The 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 to disagree about projects. Accordingly, we tried reducing 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 4 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*	

NASA10 (new NASA data up to 2010):

rank	treatment	median	IQR
1	COCONUT	34	14
1	COCOMO-II	43	35
2	TEAK	73	80
2	PEEKING2	74	51

COC05 (new COCOMO data up to 2005):

rank	treatment	median	IQR
1	COCOMO-II	46	134
1	COCONUT	62	38
1	TEAK	84	110
2	PEEKING2	87	140

NASA93 (NASA data up to 1993):

rank	treatment	median	IQR
1	COCONUT	36	38
1	COCOMO-II	39	39
1	TEAK	50	81
2	PEEKING2	65	165

COC81 (original data from the 1981 COCOMO book):

rank	treatment	median	IQR
1	COCOMO-II	32	33
1	COCONUT	33	42
2	TEAK	93	128
3	PEEKING2	131	569

Fig. 10: COCOMO vs newer methods. Displayed as per Figure 8.

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

4.3.3 Column Reduction

Prior results tell us that row reduction should be accompanied by column reduction. A study by Chen et al. [10] 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 [50] 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 $E(\sigma, a_i) = \sum_{x \in a_i} (n/N * v(r(x)))$

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 8, LOC was excluded from column reduction.

4.3.4 Results

Figure 11 compares results found when either *all* or some *reduced* set of ranges, rows, and columns are used. 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 11, 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 11 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 absolute difference between the top *r8* results and standard COCOMO is very small).

Overall, Figure 11 suggests that the modeling effort associated with COCOMO-II could be reduced. Hence, it need not be expensive to deploy parametric estimation 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 11. 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.

4.4 COCOMO with Incorrect Size Estimates

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

Before endorsing a 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?
- Was LOC measured from end-statement or end-of-line symbols?
- Or how were lines of comments handled?

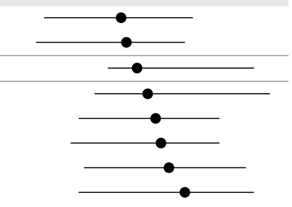
Another factor that introduces noise into training and test data are systems built from multiple languages. To make estimates from those kind of systems, KLOC in one language needs to be translated (in a possibly incorrect way) to KLOC in another language.

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.

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

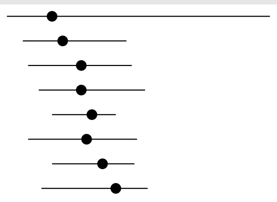
NASA10 (new NASA data up to 2010):

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



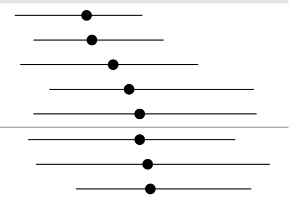
COC05 (new COCOMO data up to 2005):

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

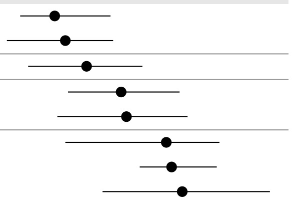


Fig. 11: COCOMO vs simpler COCOMO. Displayed as per Figure 8.

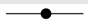
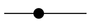
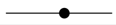
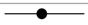
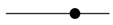
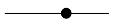
reduced to three; (2) half the columns were reduced; (3) we trained on only eight randomly selected projects; and (4) prior to train and test, all KLOC values were adjusted to

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

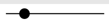
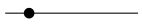
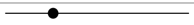
where $n \in \{0.25, 0.5\}$ is the level of noise we are exploring and r is a random number $0 \leq r \leq 1$.

In Figure 12, 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 affected by noise where the KLOC values vary up to 50% of their original value.

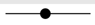
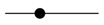
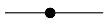
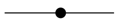
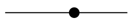
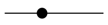
NASA10 (new NASA data up to 2010): c

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):

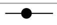
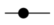
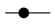
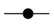
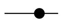
rank	treatment	median	IQR	
1	COCOMO-II	33	35	
2	COCOMO-IIn/2	47	46	
2	COCOMO-IIn/4	48	44	
2	COCONUT:c*0.5,r=8n/2	53	45	
2	COCONUT:c*0.5r=8n/4	57	44	
2	COCONUT:c*0.5,r=8	65	51	

Fig. 12: LOC noise results. MRE values. Displayed as per Figure 8.

5 Discussion

From the above, there are several open issues. Firstly, how can an estimation method based on lines of code be immune to errors in measures of those lines of code? Secondly, all the above assumes that it is possible to collect project data using the COCOMO attributes of Figure 3. What if that is not true?

5.1 KLOC Noise Immunity

At first glance, the above KLOC noise immunity result seems strange. However, it can be shown that this result can be explained by the underlying theory of COCOMO. Equation 2 commented that there is much more to effort estimation than just KLOC. Recall that Equation 2 showed how effort varies for a given KLOC. While KLOC was a factor in that equation, all the other COCOMO variables have a large influence on effort estimation. This means that even if there are errors in the KLOC measure, the other COCOMO variables can "step in" to comment on effort estimation.

Another thing to be said about Figure 12 is that that noise does degrades predicted performance to some degree (observe how the non-noisy result from COCOMO-II always have better medians than once noise is injected). So it is not true to say that KLOC noise has no effect on effort estimation. That said, it should also be noted that statistically, the size of that effect is usually very small or not significant (as defined by the statistical tests of §3.3). This could mean that our statistical tests are failing. However, looking at the large

Data set	cols	rows	median(*abs(predicted - actual) / actual*) PEEKING2	Knear(3)	CART	Knear(1)	TEAK
Albrecht	7	24	28	40 ▮▮	38 ▮▮	38 ▮▮	49 ▮▮▮▮
China	18	488	38 ▮▮	34	34	35 ▮	41 ▮▮▮▮
Cosmic	10	42	98 ▮▮▮▮	75	85 ▮▮	85 ▮▮	89 ▮▮
ISBSG10	11	37	56	62	126 ▮▮▮▮	66 ▮	65 ▮
Kemerer	7	15	42 ▮▮	24	55 ▮▮▮▮	55 ▮▮▮▮	55 ▮▮▮▮
Kitchenham	6	145	34	43 ▮▮▮	34	43 ▮▮▮	47 ▮▮▮▮
Maxwell	27	62	57 ▮▮	56 ▮▮	47	53 ▮	64 ▮▮▮▮
Miyazaki	8	48	39	41	41	57 ▮▮▮▮	57 ▮▮▮▮
Telecom	3	18	23	26 ▮▮	31 ▮▮▮▮	31 ▮▮▮▮	31 ▮▮▮▮
Usp05	7	203	30	50 ▮▮▮▮	45 ▮▮▮	40 ▮▮	50 ▮▮▮▮

KEY:

Error percentile ranges:

80 to 100 = ▮▮▮▮
60 to 80 = ▮▮▮
40 to 60 = ▮▮
20 to 40 = ▮

An absent bar denotes 0 to 20th percentile.

Percentiles computed separately for each data set.

Data set	Notes
Albrecht	Projects from IBM
China	Projects from Chinese software companies
Cosmic	Projects described in function points
ISBSG10	From the ISBSG benchmark suite
Kemerer	Large business applications
Kitchenham	Misc software projects.
Maxwell	Projects from commercial banks in Finland
Miyazaki	Japanese software projects developed in COBOL
Telecom	Maintenance projects for telecom companies
Usp05	Collected from university student projects

Fig. 13: Median errors seen in leave-one-out studies on non-COCOMO data sets. Gray bars show error discretized into 20th percentiles ranges from min to max. The PEEKING2 learner has fewest error bars; i.e. it is the best learner seen in this sample. All data available from http://openscience.us/repo/effort.

variances of the COCOMO-II results in Figure 12, another explanation might be that that large variances seen with COCOMO-II shows the inherent imprecision of effort estimation.

These large variances are an important and open issue in effort estimation. It is hard to make precise statements about effort estimation when the estimates themselves are so imprecise. Perhaps the research community should spend a few years working on variance reduction, after which, they can better explore reducing median error values.

5.2 For Non-COCOMO Data

This study has highlighted the benefits of estimating projects using COCOMO. Shepperd [63] notes that one of the disadvantages of COCOMO is that it demands projects be described in terms of the attributes of Figure 3. This is an issue since effort estimation often requires reflecting on numerous projects, some of which may no longer be active. For such historical projects, it can be difficult to uncover information such as Figure 3. Hence, it is appropriate to consider what to do for non-COCOMO projects.

Accordingly, we applied all the learners described above to the ten data sets of Figure 13. In terms of this discussion, the key feature of these data sets is COCOMO and COCONUT could not be applied to these since, apart from some effort measure, this data is described using attributes with little (if any) similarity to Figure 3.

The results of that application were not as clear cut as the results shown in Figure 8 to Figure 12. Nearly all the results had the form of Figure 14– the median results are similar and the IQR variances are large enough to make Scott-Knott declare that all learned have the same *rank=1*. This "all learners earned the same ranking" was repeated over all our non-COCOMO data sets. The reason for this "same-rank" effect was mentioned above: it is hard to make precise statements about effort estimation when the estimates themselves are imprecise. Once again, we call on the SE research community to spend more time of the variance problem in effort estimation.

Given that the quantitative statistics analysis of §3.3 could not rank learners for non-COCOMO data, we turn to a more qualitative statement. The right-hand-side columns of Figure 13 shows median errors seen in leave-one-out experiments on our ten non-COCOMO data sets. The gray bars of that figure show the median error results of a leave-one-out study

Telecom: Maintenance projects for telecommunications.

rank	treatment	median	IQR
1	PEEKING2	23	34
1	Knear(3)	26	33
1	TEAK	31	28
1	CART	31	22
1	Knear(1)	31	35

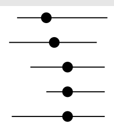


Fig. 14: Example result of applying learners to non-COCOMO data.

discretized into 20th percentiles ranges from min to max (so the *more* the gray bars, the *worse* the learner). On these non-COCOMO data sets, standard linear regression usually performed worse and PEEKING2 usually performed best. Hence, for non-COCOMO data sets we recommend PEEKING2 (but stress this qualitative recommendation is not as strong as our previous endorsement of COCOMO/COCOUNT, which can be defended via a more rigorous statistical analysis).

Why is PEEKING2 best for non-COCOMO data and not for COCOMO data? The answer, we conjecture, is that model-based effort estimation involves *data collection* followed by *model generation*. Perhaps if the data collection is very broad and covers all aspects of a project such as the process, product, personnel, platform information collected by COCOMO, then the subsequent model generation can be very simple (just the COCOMO equation of Figure 4). On the other hand, if data collection is more myopic and focuses on just the data that is serendipitously available, then the subsequent model generation task is more challenging. For such challenging tasks, we would expect:

- The generator needs to be sophisticated; e.g. not something as simplistic as the linear regression used in Figure 13 (hence, PEEKING2 did well in that figure);
- No matter how sophisticated the model generator is, it may not perform as well as models generated from a broader range of data (hence, PEEKING2 performed worse than COCOMO in Figure 10).

6 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.

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.

7 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.

Based on that study, we offered a negative result in which a decades old effort estimation method performed as well, or better, as more recent methods:

- **RQ1**: just using LOC for estimation is far worse that parametric estimation over many attributes (see §4.1);

– **RQ2**: new innovations in effort estimation have not superseded parametric estimation (see §4.2);
– **RQ3**: Old parametric tunings are not out-dated (see §4.2);
– **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 §4.3);
– **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 §4.4);.

Hence, we conclude that in 2016, it is still a valid and a recommended practice to *first* try parametric estimation. This conclusions comes with certain caveats:

– It can sometimes be useful to augment standard COCOMO with a local calibration method like COCONUT and column pruner of §4.3.3. Specifically, using COCONUT (plus some column reduction), we found that adequate estimates can be generated using just a handful of prior projects. In these experiments, eight projects were enough for COCONUT (and we are exploring methods to reduce that even further). This is an important result since, given the rapid pace of change on software engineering, it is unlikely organizations will have access to dozens and dozens of prior relevant projects to learn from.
– For project data that does *not* contain the COCOMO attributes of Figure 3. then there is some evidence that more recent estimation methods are useful. However, as shown in §5.2, we found it difficult to make a rigorous statistical case than learnerX was better than learnerY.

Our take-away message here is that the choice of data to collect may be more important than what learner is applied to that data. Certainly, it is true that not all projects can be expressed in terms of COCOMO. But when there is a choice, we recommend collecting data like Figure 3, and then processing that data using COCOMO-II.

8 Future Work

The negative results of this paper makes us question some of the newer (and supposedly better) innovative techniques for effort estimation. The unique and highly variable characteristics of SE project data place great limitation on the results obtained by naively applying some brand-new algorithm. Perhaps one direction for future direction is to investigate how innovative new techniques can extend (rather than replace) existing and successful estimation methods.

Having endorsed the use of parametric methods such as COCOMO, it is approprite to discuss current plans for new versions of that approach. 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.

Additionally, while calibrating COCOMO II model and developing COCOMO III, we were also seeing time-competitive Agile projects in well-jelled, domain-experienced rapid development organizations, which demonstrates tremendous effort reduction and schedule acceleration [21]. Finally, the emerging community-based software development, i.e. software crowd sourcing [26], challenges the underlying assumptions of traditional software estimation laws. Access to external workforce and competition factors are becoming critical development influential factors and need to be further investigated.

Efforts to characterize these models and to gather data to calibrate models for dealing with them are underway. Contributors to the definition and calibration are most welcome.

Acknowledgements

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. 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.
2. 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.
3. 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`.
4. 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.
5. B. Boehm. *Software Engineering Economics*. Prentice Hall, 1981.
6. B. Boehm. Safe and simple software cost analysis. *IEEE Software*, pages 14–17, September/October 2000.
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. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. 1984.
9. 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.
10. Zhihao Chen, Barry Boehm, Tim Menzies, and Daniel Port. Finding the right data for software cost modeling. *IEEE Software*, 22:38–46, 2005.
11. 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`.
12. 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.
13. P R Cohen. *Empirical Methods for Artificial Intelligence*. MIT Press, 1995.
14. 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.
15. R. Cordero, M. Costamagna, and E. Paschetta. A genetic algorithm approach for the calibration of cocomo-like models. In *12th COCOMO Forum*, 1997.
16. 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`.
17. 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.
18. Bradley Efron and Robert J Tibshirani. *An introduction to the bootstrap*. Mono. Stat. Appl. Probab. Chapman and Hall, London, 1993.
19. 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.
20. 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.
21. Dan Ingold, Barry Boehm, and Supannika Koolmanojwong. A model for estimating agile project process and schedule acceleration. In *ICSSP 2013*, pages 29–35.
22. R. Jensen. An improved macrolevel software development resource estimation model. In *5th ISPA Conference*, pages 88–92, April 1983.

23. 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.
24. 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`.
25. Magne Jorgensen. A review of studies on expert estimation of software development effort. *Journal of Systems and Software*, 70(1-2):37–60, February 2004.
26. M. Li K. Mao, Y. Yang and M. Harman. Pricing crowdsourcing-based software development tasks. In *ICSE, New Ideas and Emerging Results*, pages 1205–1208, San Francisco, CA, USA, 2013.
27. G. Kadoda, M. Cartwright, L. Chen, and M. Shepperd. Experiences using casebased reasoning to predict software project effort, 2000.
28. 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.
29. 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.
30. 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.
31. 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.
32. C. Kirsopp and M. Shepperd. Making inferences with small numbers of training sets. *IEEE Proc.*, 149, 2002.
33. 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`.
34. E. Kocaguneli, T. Menzies, and J.W. Keung. On the value of ensemble effort estimation. *Software Engineering, IEEE Transactions on*, 38(6):1403–1416, Nov 2012.
35. 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.
36. Ekrem Kocaguneli, Tim Menzies, and Emilia Mendes. Transfer learning in effort estimation. *Empirical Software Engineering*, pages 1–31, 2014.
37. Ekrem Kocaguneli, Thomas Zimmermann, Christian Bird, Nachiappan Nagappan, and Tim Menzies. Distributed development considered harmful? In *ICSE*, pages 882–890, 2013.
38. 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.
39. 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.
40. 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.
41. 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.
42. 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.
43. 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.
44. 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`.
45. 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`.
46. Tim Menzies, Alex Dekhtyar, Justin Distefano, and Jeremy Greenwald. Problems with Precision. *IEEE Transactions on Software Engineering*, September 2007.

47. Tim Menzies, Fayola Peters, and Andrian Marcus. Ooops... (errata report for "Better Cross-Company Learning"). In *MSR'13*, 2013. http://www.slideshare.net/timmenzies/msr13-mistake.
48. 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`.
49. Tim Menzies and Martin Shepperd. Special issue on repeatable results in software engineering prediction. *Empirical Software Engineering*, 17(1-2):1–17, 2012.
50. A. Miller. *Subset Selection in Regression (second edition)*. Chapman & Hall, 2002.
51. 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.
52. 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.
53. 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.
54. Emerson Murphy-Hill, Chris Parnin, and Andrew P. Black. How We Refactor, and How We Know It. *IEEE Transactions on Software Engineering*, 38(1):5–18, 2012.
55. 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.
56. Vasil Papakroni. Data carving: Identifying and removing irrelevancies in the data. Master's thesis, Lane Department of Computer Science and Electrical Engineering, West Virginia Unviersity, 2013.
57. R. Park. The central equations of the price software cost model. In *4th COCOMO Users Group Meeting*, November 1988.
58. Carol Passos, Ana Paula Braun, Daniela S. Cruzes, and Manoel Mendonca. Analyzing the impact of beliefs in software project practices. In *ESEM'11*, 2011.
59. K R Popper. *Conjectures and Refutations,*. Routledge and Kegan Paul, 1963.
60. D. Posnett, V. Filkov, and P. Devanbu. Ecological inference in empirical software engineering. In *Proceedings of ASE'11*, 2011.
61. L. Putnam. A macro-estimating methodology for software development. In *Proceedings, IEEE COMPCON76 Fall*, pages 38–43, September 1976.
62. 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.
63. M. Shepperd. Software project economics: A roadmap. In *International Conference on Software Engineering 2007: Future of Software Engineering*, 2007.
64. 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`.
65. Martin J. Shepperd and Steven G. MacDonell. Evaluating prediction systems in software project estimation. *Information & Software Technology*, 54(8):820–827, 2012.
66. Spareref.com. Nasa to shut down checkout & launch control system, August 26, 2002. `http://www.spaceref.com/news/viewnews.html?id=475`.
67. R. Valerdi. Convergence of expert opinion via the wideband delphi method: An application in cost estimation models. In *Incose International Symposium, Denver, USA*, 2011. Available from http://goo.gl/Zo9HT.
68. Fiona Walkerden and Ross Jeffery. An empirical study of analogy-based software effort estimation. *Empirical Softw. Engg.*, 4(2):135–158, 1999.
69. C. Walston and C. Felix. A method of programming measurement and estimation. *IBM Systems Journal*, (1):54–77, 1977.
70. R. Wolverton. The cost of developing large-scale software. *IEEE Trans. Computers*, pages 615–636, June 1974.