Empir Software Eng (2017) 22:2658–2683
DOI 10.1007/s10664-016-9472-2

Negative results for software effort estimation

Tim Menzies[1] · **Ye Yang**[2] · **George Mathew**[1] ·
Barry Boehm[3] · **Jairus Hihn**[4]

Published online: 21 November 2016

Abstract 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. Hence, this paper checks if new SEE methods generated better estimates than older methods, as follows. *First*, we 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, and a recent baseline method proposed in ACM Transactions on Software Engineering). *Second*, we catalog the list of objections that lead to the development of post-COCOMO estimation methods. *Third*, we characterize each of those objections as a comparison between newer and older estimation methods. *Fourth*,

Communicated by: Richard Paige, Jordi Cabot and Neil Ernst

✉ Tim Menzies
tim.menzies@gmail.com

Ye Yang
ye.yang@stevens.edu

George Mathew
george.meg91@gmail.com

Barry Boehm
barryboehm@gmail.com

Jairus Hihn
jairus.hihn@jpl.nasa.gov

[1] CS, North Carolina State University, Raleigh, NC, USA

[2] SSE, Stevens Institute, Hoboken, NJ, USA

[3] CS, University of Southern California, Los Angeles, CA, USA

[4] Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, USA

using four COCOMO-style data sets (from 1991, 2000, 2005, 2010) we run those comparisons experiments. *Fifth*, we compare the performance of the different estimators using a Scott-Knott procedure (that includes an A12 effect size to rule out "small differences and a 99 % confident bootstrap procedure to check for statistically different groupings of treatments). The major negative results of this paper are that for the COCOMO data sets, nothing we studied did any better than Boehms original procedure. When COCOMO-style attributes are available, we strongly recommend (i) using that data and (ii) use COCOMO to generate predictions. We say this 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*.

Keywords Effort estimation · COCOMO · CART · Nearest neighbor · Clustering · Feature selection · Prototype generation · Bootstrap sampling · Effect size · A12

1 Introduction

This paper is about a negative result in software effort estimation– specifically:

- For project data expressed in a certain way (the COCOMO format (Boehm et al. 2000));
- Despite decades of work into alternate methods;
- Best predictions from that data come from a parametric method proposed in 2000 (Boehm et al. 2000).

This conclusion comes with two caveats. Firstly, not all projects can be expressed in terms of COCOMO– but when there is a choice, the results of this paper argue that there is value in using that format. Secondly, our conclusion is about *solo* prediction methods which is different to the *ensemble* approach (Minku and Yao 2011, 2013; Kocaguneli et al. 2012; Menzies et al. 2015)– but if using ensembles, this paper shows that parametric estimation would be a viable ensemble member.

For pragmatic and methodological reasons,it is important to report negative results like the one described above. 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 (Popper 1963)), 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 (Menzies et al. 2007)), 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 negative results in the field of software analytics. What does happen, occasionally, are reports of small corrections to prior work. Given the complexity of software analytics, this absence of such failure reports is highly suspicious. For examples of such reports, see (e.g. as done in Menzies et al. 2013; Murphy-Hill et al. 2012).

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 (1995) strongly advises such "straw man" comparisons, since sometimes, they reveal that the supposedly superior method is actually overly complex. Hence it always useful to compare methods against simpler alternative.

That said, in some cases no such method is available making such benchmarking impossible. Although as a domain starts to become more mature, these comparisons can be conducted; see, e.g. the many experiments on defect prediction (Scanniello et al. 2013) or tag inference for Stack Overflow posts (Stanley and Byrne 2013). Accordingly, this paper checks an interesting, but currently unexplored aspect of effort estimation. We check if there exists data sets from which very old methods do just as well as anything else. We consider data expressed in terms of the COCOMO ontology: 23 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 four 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).

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? COCOMO models are learned by "tuning" the default model parameters using local project data. COCOMO-II shipped with a set of parameters learned from a particular set of projects from 1995 to 2000. We apply those COCOMO-II tunings, without modification, to a wide range of projects dating from 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.

To explore these questions, we use COCOMO since its internal details have been fully published (Boehm et al. 2000). Also, we can access a full implementation of the 2000 COCOMO model. Further, we have access to numerous interesting COCOMO data sets: see Figs. 1 and 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 Fig. 2.

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

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	

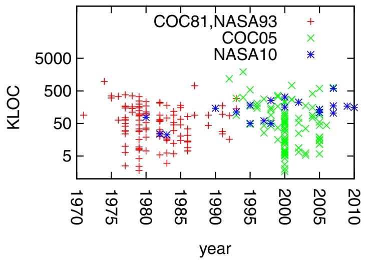


Fig. 1 Projects used by the learners in this study. Figure 4 shows project attributes. COC81 is the original data from 1981 COCOMO book (Boehm 1981). This comes from projects dating 1970 to 1980. NASA93 is NASA data collected in the early 1990s about software that supported the planning activities for the International Space Station. Our other data sets are NASA10 and COC05 (the latter is proprietary and cannot be released to the research community). The non-proprietary data (COC81 and NASA93 and NASA10) is available at http://openscience.us/repo or in Fig. 3

Using that data, the experiments of this paper conclude that the answer to all our research questions is nearly always "no". The RQ1 experiments show that good estimates use many variables and poorer estimates result from some trite calculation based on KLOC. As to the other research questions (RQ2, RQ3, RQ4), those results imply that the continued use of parametric estimation can still be endorsed– at least for data expressed in terms of the 23 COCOMO attributes.

For a sample of our data see the NASA10 data set in Fig. 3.

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 (Spareref.com 2002). Over-estimation delays funding to other promising ideas and organizational competitiveness (Kocaguneli et al. 2012). Hence, there is a long history of researchers exploring software effort estimation; e.g. Wolverton (1974), Freiman and Park (1979), Putnam (1976), Black et al. (1977), Herd et al. (1977), Walston and Felix (1977), Jensen (1983), Park (1988), Boehm (1981), Walkerden and Jeffery (1999), Shepperd and Schofield (1997),

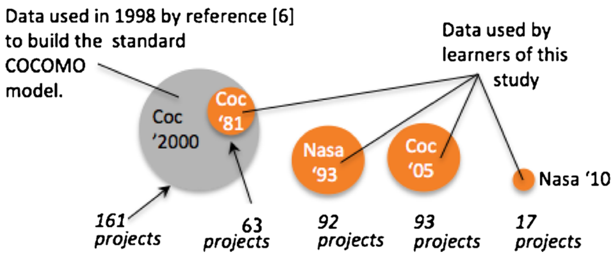

Data used in 1998 by reference [6] to build the standard COCOMO model.
Data used by learners of this study
Coc '2000
Coc '81
Nasa '93
Coc '05
Nasa '10
161 projects
63 projects
92 projects
93 projects
17 projects

Fig. 2 Projects in this study. COC81 is a subset of COCOMO-II. Note that NASA'93 and COC'05 and NASA'10 have no overlap with the data used to define the version of COCOMO used in this paper

prec	flex	resl	team	pmat	rely	cplx	data	ruse	time	stor	pvol	acap	pcap	pcon	aexp	plex	ltex	tool	sced	site	docu	kloc	months
2	2	2	3	3	4	5	4	3	5	6	4	4	4	3	4	3	3	1	3	4	4	77	1830
2	2	2	3	3	5	5	2	3	5	6	2	4	3	3	2	1	2	2	3	4	4	24	648
2	2	2	3	3	4	5	3	3	5	5	4	3	3	3	3	2	2	1	3	4	4	23	492
2	2	3	3	2	4	4	3	2	3	3	4	3	3	3	3	3	4	2	3	5	3	146	3292
2	3	3	5	3	3	4	3	2	4	4	2	5	5	4	5	1	5	3	3	6	3	113	1080
3	3	3	3	3	3	4	3	2	3	3	3	3	3	3	4	3	4	2	3	4	3	184	1043
5	3	3	3	4	4	4	3	2	3	3	2	3	3	3	5	3	4	2	3	5	3	61	336
5	3	3	4	4	4	5	3	2	3	3	2	3	3	3	5	3	4	2	3	6	3	50	637
3	3	3	2	3	4	5	3	2	3	3	3	3	3	3	4	3	4	2	3	5	3	253	2519
3	3	4	3	3	4	4	3	4	3	3	2	3	4	3	3	1	4	5	3	2	3	159	1048
3	3	3	3	3	4	5	3	2	3	3	4	4	4	5	4	4	4	2	1	5	3	324	1735
3	2	4	4	3	4	5	3	4	3	4	5	4	4	3	4	4	3	4	2	6	3	224	691
5	2	2	4	3	4	3	3	4	5	4	3	4	4	3	4	4	4	3	3	3	3	105	320
3	2	2	4	3	4	3	3	3	3	3	2	4	4	3	4	4	4	3	3	3	3	173	329
3	2	4	3	3	4	5	3	4	3	3	4	3	4	4	4	3	3	3	3	5	3	597	1705
4	2	4	3	5	4	3	2	3	3	4	4	2	2	3	3	5	5	3	3	5	3	155	789
4	3	3	3	4	4	4	3	2	3	3	3	4	4	3	5	4	4	2	3	5	3	170	552
scale factors					effort multipliers																	size	effort

Fig. 3 The 17 projects in NASA10 (one row per project). For a definition of the terms in row1 ("prec", "flex", "resl" etc.) see Fig. 4. As to the different columns, scale factors change effort exponentially while effort multipliers have a linear impact on effort. Any effort multiplier with a value of "3" is a *nominal* value; i.e. it multiplies the effort by a multiple of 1.0. Effort multipliers above and below "3" can each effect project effort by a multiple ranging from 0.7 to 1.74. For full details on how these values are used, see Fig. 5

Jørgensen and Shepperd (2007), Menzies et al. (2006), and Burgess and Lefley (2001). 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 (Jørgensen and Shepperd 2007). Since then, many such papers have been published; e.g. Lokan and Mendes (2006), Corazza et al. (2010), Minku and Yao (2014), Li and Ruhe (2007), Li et al. (2009), Keung (2008), Keung et al. (2008), Keung and Kitchenham (2008), Kocaguneli et al. (2012), Menzies and Shepperd (2012), Kocaguneli et al. (2013), and Kocaguneli et al. (2014).

In the 1970s and 1980s, this kind of research was focused on *parametric estimation* as done by Putnam and others (Wolverton 1974; Freiman and Park 1979; Black et al. 1977; Herd et al. 1977; Walston and Felix 1977; Boehm 1981). For example, Boehm's COnstructive COst MOdel (COCOMO) model (Boehm 1981) 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, with minor tweaks (Menzies et al. 2005). 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 (Shepperd and Schofield 1997) case-based-reasoning (Shepperd and Schofield 1997), spectral clustering (Menzies et al. 2013), genetic algorithms (Cordero et al. 1997; Burgess and Lefley 2001), etc. These methods can be augmented with "meta-level" techniques like tabu search (Corazza et al. 2010), feature selection (Chen et al. 2005), instance selection (Kocaguneli et al. 2012), feature synthesis (Menzies and Shepperd 2012), active learning (Kocaguneli et al. 2013), transfer learning (Kocaguneli et al. 2014). temporal learning (Lokan and Mendes 2009; Minku and Yao 2014), and many more besides (Fig. 4).

	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 cooperative	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
Effort				
months	construction effort in months	1 month = 152 hours (includes development & management hours).		

Fig. 4 COCOMO-II attributes

2.2 Current Practice

In her keynote address to ICSE'01, Mary Shaw (Shaw 2001) 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 (Shepperd and Schofield 1997) or case-based-reasoning (Shepperd and Schofield 1997). 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 (Dabney 2002).
- 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 http://tiny.cc/iceaa, http://tiny.cc/nasa_cost, http://tiny.cc/csse).

2.3 But Does Anyone Use COCOMO?

Two of the myths of effort estimation is that (1) no one uses model-based estimation like COCOMO; and (2) estimates are always better done using expert-based guess-timation.

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 expert-based estimation is a common practice (Boehm 2000), this is not to say that this should be recommended as the *best* or *only* way to make estimates:

- Jorgensen (2004) reviews studies comparing model- and expert- based estimation and concludes that there there is no clear case that expert-methods are better.
- In 2015, Jorgensen further argued (Jorgensen 2015) that model-based methods are useful for learning the *uncertainty* about particular estimates; e.g. by running those models many times, each time applying small mutations to the input data.
- Valerdi (2011) lists the cognitive biases that can make an expert offer poor expert-estimates.
- Passos et al. show that many commercial software engineers generalize from their first few projects for all future projects (Passos et al. 2011).
- Jørgensen and Gruschke (2009) document how commercial "gurus" rarely use lessons from past projects to improve their future expert-estimates. They offer examples where this failure to revise prior beliefs leads to poor expert-based estimates.

Much research has concluded that the best estimations come from *combining* the predictions from *multiple oracles* (Kocaguneli et al. 2012; Chulani et al. 1999; Baker 2007; Valerdi 2011). Note that it is much easier to apply this double-check strategy using expert+model-based methods than by comparing the estimates from multiple expert teams. For example, all the model-based methods studied in this paper can generate estimates in just a few seconds. In comparison, expert-based estimation is orders of magnitude slower– as seen in Valerdi's COSYSMO expert-method. While a strong proponent of this approach, Valerdi concedes that "(it is) extremely time consuming when large sample sizes are needed" (Valerdi 2011). For example, he once recruited 40 experts to three expert 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 expert-based method. An alternate, more lightweight expert-method is "planning poker" (Molokken-Pstvold et al. 2008) 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 are 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 Molokken-Pstvold et al. 2008). Also, 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 expert-based estimation 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 (Wolverton 1974), 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 expert exercises to find the relative significance of various cost drivers. Combining local expertise and data, plus some prior results such as Putnam (1976), Black et al. (1977), Herd et al. (1977), and Walston and Felix (1977), and early versions of the RCA PRICE S model (Freiman and Park 1979), 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 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 (Boehm 1981). 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 Fig. 5 that map the project descriptors (very low, low, etc) into the specific values used in the COCOMO-II model (released in 2000 (Boehm et al. 2000)):

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

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.

```
_   = 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, 5 ,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. 5 COCOMO-II: effort estimates from a *project*. Here, *project* has 5 scale factors plus 17 effort multipliers plus KLOC. "Xhigh" is show for "extremely high". Each attribute except KLOC and effort is scored using the scale very low = 1, low = 2, up to xhigh = 6. Note all attributes extend across the entire range very low to extremely high since, in Boehm's modeling work, not all effects extend across the entire range. For an explanation of the attributes shown in green, see Fig. 4

2.5 COCOMO and Local Calibration

COCOMO models are learned by "tuning" the default model parameters using local project data. When local data is scarce, approximations can be used to tune a model using just a handful of examples.

For example, COCOMO's *local calibration* procedure, adjusts the impact of the scale factors and effort multipliers by tuning the a, b values of (1) while keeping the other values of the tuning matrix constant as shown in Fig. 5. 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 Fig. 6 (first written in 2002 and first published in 2005 (Menzies et al. 2005)). 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,

```
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. 6 COCONUT tunes a, b of Fig. 5's COCOMO function

by 2/3rds). 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 research 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" (Posnett et al. 2011; Menzies et al. 2013). To avoid ecological inference, our rig in Fig. 7 runs separately for each data set.

Since some of our methods include a stochastic algorithm (the COCONUT algorithm of Fig. 6), 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 Fig. 7 is a "leave-one-out experiment"; i.e. training is conducted on all-but-one example, then tested on the "holdout" example not seen in training. This separation of training and testing data is of particular importance in this study. As shown in Fig. 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 Fig. 4), 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.

We assess performance via *Standardized Error(SE)*; i.e.

$$SE = \frac{\sum_{i=1}^{n}(abs(actual_i - predicted_i))/n}{\sum_{i=1}^{n}(abs(actual_i - sampled))/n} * 100 \tag{2}$$

```
def RIG():

 DATA = { COC81, NASA83, COC05, NASA10 }

 for data in DATA: # e.g. data = COC81

     errors= {}
     for learner in LEARNERS: #e.g. learner=COCONUT
       for n in range(10): # ten 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)
           error    = abs(actual - estimate)/actual
           errors[learner][n] = error

     print rank(errors) # some statistical tests
```

Fig. 7 The experimental rig used in this paper

This measure is derived from *Standardized Accuracy*(SA) defined by Shepperd & MacDonnell Shepperd and Macdonell (2012) ($SE = 1 - SA$). In (2), $actual$ represents the true value, $predicted$ represents the estimated value by the predictor and $sampled$ is a value drawn randomly from a list of random samples(with replacement) from the training data. Shepperd and MacDonnell also 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).

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:

$$dist(x, y) = \sqrt{\sum_i w_i (x_i - y_i)^2} \tag{3}$$

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 (1999); e.g.. for $loc(3)$, the estimate from the first, second and third closest neighbor with estimates a, b and c respectively is

$$effort = (3a + 2b + 1c)/6 \tag{4}$$

We also baseline the COCOMO-II and COCONUT models using the *Automatically Transformed Linear Baseline Model*(ATLM) proposed by Whigham et al. (2015). ATLM is a multiple linear regression model of the form

$$effort_i = \beta_0 + \beta_1.x_{1i} + \beta_2.x_{2i} + \ldots + \beta_n.x_{ni} + \epsilon_i \tag{5}$$

where $effort_i$ is the quantitative response(effort) for project i and x_i are the independent variables of describing the project. The prediction weights β_i are determined using a least square error estimation. Transformations are also employed on the independent variables(x_i) based on their nature. If the variable is continuous in nature, either a logarithmic, a square root transformation or no transformation is employed such that the skewness of the independent variable in the training set is minimized. If the variable is of categorical nature, no transformation is performed on the model.

Apart from the LOC "straw man" and the ATLM baseline we also compare COCOMO-II and COCONUT with CART and Knear(n) as they proved their value in the 1990s (Shepperd and Schofield 1997; Walkerden and Jeffery 1999). That said, CART and Knear(n) still have currency: recent results from IEEE TSE 2008 and 2012 still endorse their use for effort estimation (Dejaeger et al. 2012; Kocaguneli et al. 2012; Keung et al. 2008). 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. Further, to account for some of the more recent work on effort estimation, we also use TEAK and PEEKING2 (Kocaguneli et al. 2012; Papakroni 2013).

CART (Breiman et al. 1984) 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 (Shepperd and Schofield 1997). 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 (4). Knear(n) is an example of

CBR; i.e. *case-based reasoning*. CBR for effort estimation was first pioneered by Shepperd & Schofield in 1997 Shepperd and Schofield (1997). Since then, it has been used extensively in software effort estimation (Auer et al. 2006; Walkerden and Jeffery 1999; Kirsopp and Shepperd 2002; Shepperd and Schofield 1997; Kadoda et al. 2000; Li and Ruhe 2006, 2007, 2008; Li et al. 2009; Keung 2008; Keung et al. 2008; Keung and Kitchenham 2008). There are several reasons for this. Firstly, it works even if the domain data is sparse (Myrtveit et al. 2005). 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 (Kocaguneli et al. 2012). 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 (Papakroni 2013) 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 Fig. 8. One important detail with TEAK and PEEKING2 is that when they prune data, they only do so on the *training* data. Given a test set, TEAK and PEEKING2 will always try to generate estimates for all members of that test set.

3.3 Choice of Statistical Ranking Methods

The last line of our experimental rig shown in Fig. 7 *rank*s multiple methods for learning effort estimators. For this paper, those multiple methods are the range of *l* treatments of size *ls* = | *l* | explored within each research question. For example, *RQ1* studies the differences in output produced by *ls* = 4 methods: two COCOMO variants and two others that just use lines of code counts.

This study ranks methods using the Scott-Knott procedure recommended by Mittas & Angelis in their 2013 IEEE TSE paper (Mittas and Angelis 2013). This method sorts a list of *l* treatments with *ls* measurements by their median score. It then splits *l* into sub-lists *m*, *n*

- PEEKING2's feature weighting scheme changes w_i in Equation 3 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 3 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. 8 Inside PEEKING2 (Papakroni 2013)

in order to maximize the expected value of differences in the observed performances before and after divisions. For example, for **RQ1**, we would sort $ls = 4$ methods based on their median score, then divide them into three sub-lists of of size $ms, ns \in \{(1, 3), (2, 2), (3, 1)\}$. Scott-Knott would declare one of these divisions to be "best" as follows. For lists l, m, n of size ls, ms, ns where $l = m \cup n$, the "best" division maximizes $E(\Delta)$; i.e. the difference in the expected mean value before and after the spit:

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

Scott-Knott then checks if that "best" division is actually useful. To implement that check, Scott-Knott would apply some statistical hypothesis test H to check if m, n are significantly different. If so, Scott-Knott then recurses on each half of the "best" division.

For a more specific example, consider the results from $l = 5$ treatments:

```
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 ($A12 \geq 0.6$).

For a justification of the use of non-parametric bootstrapping, see Efron and Tibshirani (1993, p220–223). For a justification of the use of effect size tests see Shepperd and Macdonell (2012), Kampenes et al. (2007), and Kocaguneli et al. (2013). 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:

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

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 (Arcuri and Briand 2011).

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 Fig. 9, this is not necessarily true. This figure is a comparative ranking for LOC(1) LOC(3), COCOMO-II and COCONUT. The rows of Fig. 9 are sorted by the SE figures. These rows are divided according to their *rank*, shown in the left column: better methods have *lower rank* since they have *lower SE* 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 Fig. 9 is that just using lines of code is *not* better than parametric estimation. If the reader is surprised by this result, then we note that with a little mathematics,

NASA10 (new NASA data up to 2010):

rank	treatment	median	IQR
1	COCOMO-II	37	57
1	COCONUT	39	54
1	loc(3)	47	93
1	loc(1)	75	98

COC05 (new COCOMO data up to 2005):

rank	treatment	median	IQR
1	COCOMO-II	12	52
2	loc(1)	21	56
2	loc(3)	22	55
2	COCONUT	22	89

NASA93 (NASA data up to 1993):

rank	treatment	median	IQR
1	COCONUT	12	48
1	COCOMO-II	15	50
2	loc(1)	23	63
2	loc(3)	35	65

COC81 (original data from the 1981 COCOMO book):

rank	treatment	median	IQR
1	COCOMO-II	3	21
1	COCONUT	4	24
2	loc(3)	14	36
2	loc(1)	19	42

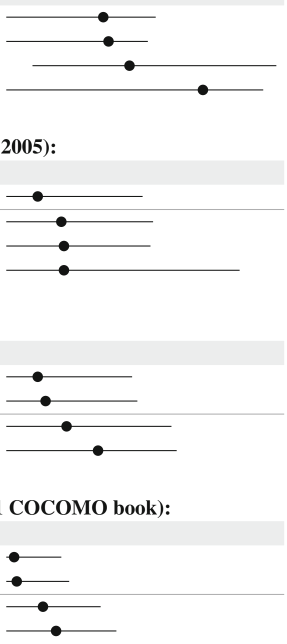


Fig. 9 COCOMO vs just lines of code. SE values seen in leave-one-studies, repeated ten times. For each of the four tables in this figure, *better* methods appear *higher* in the tables. In these tables, median and IQR are the 50th and the (75–25)th percentiles. The IQR range is shown in the right column with *black dot* at the median. *Horizontal lines* divide the "ranks" found by our Scott-Knott+bootstrapping+effect size tests (shown in *left column*)

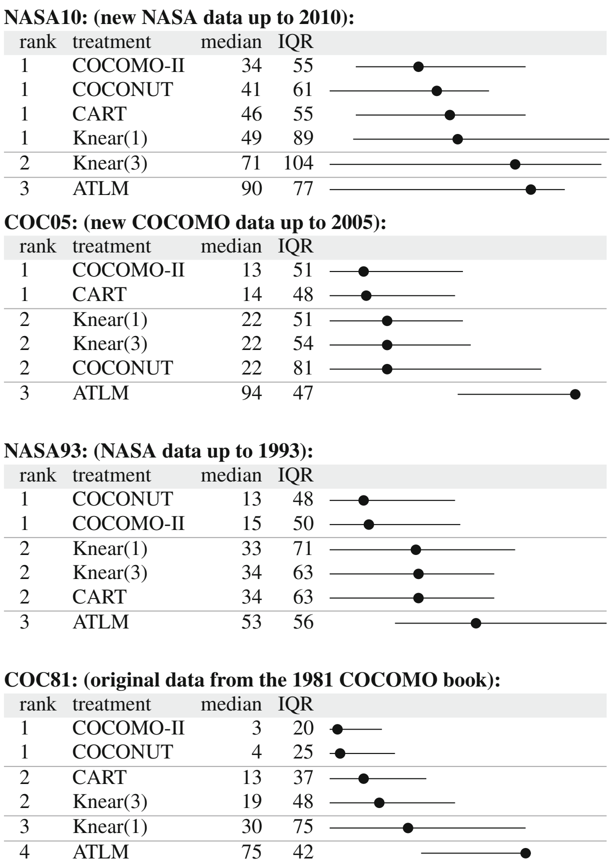
it is possible to show that the results of Fig. 9 are not surprising. From (1), recall that the minimum effort is bounded by the *sum* of the minimum scale factors and the *product* of the minimum effort multipliers. Similar expressions hold for the maximum effort estimate. Hence, for a given KLOC, the range of values is given by:

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

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

$$154/0.18 * KLOC^{1.23-0.97} = 856 * KLOC^{0.25} \tag{6}$$

Equation (6) 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.



NASA10: (new NASA data up to 2010):

rank	treatment	median	IQR
1	COCOMO-II	34	55
1	COCONUT	41	61
1	CART	46	55
1	Knear(1)	49	89
2	Knear(3)	71	104
3	ATLM	90	77

COC05: (new COCOMO data up to 2005):

rank	treatment	median	IQR
1	COCOMO-II	13	51
1	CART	14	48
2	Knear(1)	22	51
2	Knear(3)	22	54
2	COCONUT	22	81
3	ATLM	94	47

NASA93: (NASA data up to 1993):

rank	treatment	median	IQR
1	COCONUT	13	48
1	COCOMO-II	15	50
2	Knear(1)	33	71
2	Knear(3)	34	63
2	CART	34	63
3	ATLM	53	56

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

rank	treatment	median	IQR
1	COCOMO-II	3	20
1	COCONUT	4	25
2	CART	13	37
2	Knear(3)	19	48
3	Knear(1)	30	75
4	ATLM	75	42

Fig. 10 COCOMO vs standard methods. Displayed as per Fig. 9

4.2 COCOMO vs Other Methods

This section explores **RQ2: Has parametric estimation been superseded by more recent estimation methods?** and **RQ3: Are the old parametric tunings irrelevant to more recent projects?**

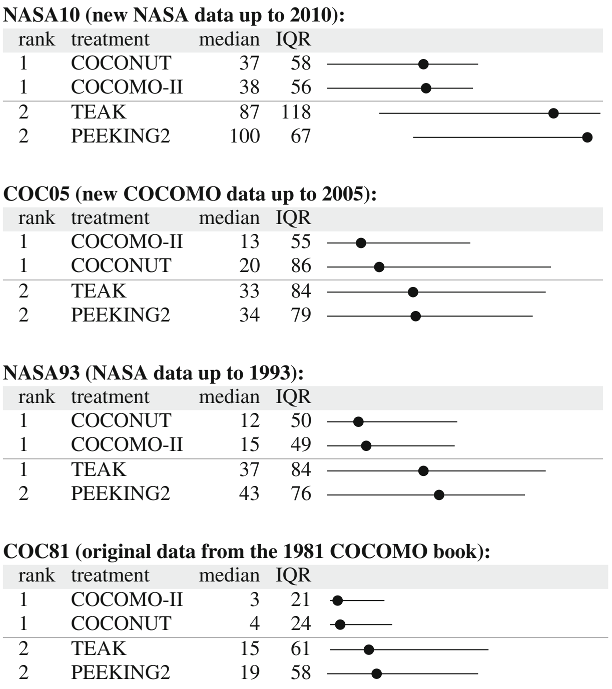
Figure 10 compares COCOMO and COCONUT with standard effort estimation methods from the 1990s (CART and Knear(n)) as well as ATLM (the baseline effort estimation method proposed in 2015 by Whigham et al. (2015) (a method defined by its authors to better define effort estimation experiments– and perhaps to encourage more repeatability in these kinds of studies). In that comparison, COCOMO-II's error is not ranked worse than any other method (sometimes COCONUT had a slightly lower median SE but that difference was small: $\leq$2 %).

Figure 11 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 often ranked equaled to COCOMO-II. In several cases COCOMO-II and COCONUT were ranked first and second and the median difference in their scores is very small.

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 2000



NASA10 (new NASA data up to 2010):

rank	treatment	median	IQR
1	COCONUT	37	58
1	COCOMO-II	38	56
2	TEAK	87	118
2	PEEKING2	100	67

COC05 (new COCOMO data up to 2005):

rank	treatment	median	IQR
1	COCOMO-II	13	55
1	COCONUT	20	86
2	TEAK	33	84
2	PEEKING2	34	79

NASA93 (NASA data up to 1993):

rank	treatment	median	IQR
1	COCONUT	12	50
1	COCOMO-II	15	49
1	TEAK	37	84
2	PEEKING2	43	76

COC81 (original data from the 1981 COCOMO book):

rank	treatment	median	IQR
1	COCOMO-II	3	21
1	COCONUT	4	24
2	TEAK	15	61
2	PEEKING2	19	58

Fig. 11 COCOMO vs newer methods. Displayed as per Fig. 9

(all of COC81, plus some of NASA93) but also for projects completed up to a decade after those tunings (NASA10).

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 Fig. 5 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*		

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. (2005) 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 (2002) 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 Fig. 9, LOC was excluded from column reduction.

4.3.4 Results

Figure 12 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.

Figure 12 suggests that it is defensible to learn a COCOMO model from just four to eight projects. Most of 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: just 2 %).

Overall, Fig. 12 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, a mere four to 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.

5 Threats to Validity

Questions of validity 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.

To increase external validity, all the data used in this work is available on-line in the PROMISE code repository. Also, our use of a leave-one-out experimental rig plus the public availability of three of our four data sets (NASA93, COC81, NASA10) means that other researchers would be able to reproduce exactly our rig on exactly the code used in this study.

One 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. Any case studies in SE data mining can only explore a small subset of options, selected by the biases of the researcher. The best any researcher can hope to do is state their biases and make some attempt to compensate for them. Accordingly:

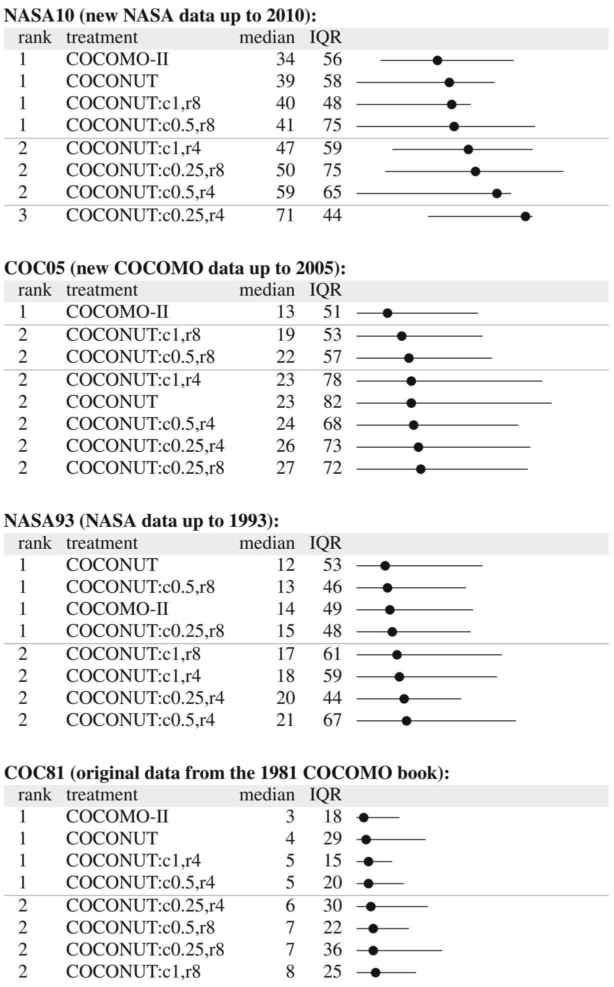
– The biases of the authors of this paper made us select a parametric modeling method (COCOMO) as the main modeling method.

– We then made a conscious decision to reverse those biases and explore non-parametric methods (PEEKING2 and TEAK) as well as decision-tree methods (CART).

6 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:



NASA10 (new NASA data up to 2010):

rank	treatment	median	IQR
1	COCOMO-II	34	56
1	COCONUT	39	58
1	COCONUT:c1,r8	40	48
1	COCONUT:c0.5,r8	41	75
2	COCONUT:c1,r4	47	59
2	COCONUT:c0.25,r8	50	75
2	COCONUT:c0.5,r4	59	65
3	COCONUT:c0.25,r4	71	44

COC05 (new COCOMO data up to 2005):

rank	treatment	median	IQR
1	COCOMO-II	13	51
2	COCONUT:c1,r8	19	53
2	COCONUT:c0.5,r8	22	57
2	COCONUT:c1,r4	23	78
2	COCONUT	23	82
2	COCONUT:c0.5,r4	24	68
2	COCONUT:c0.25,r4	26	73
2	COCONUT:c0.25,r8	27	72

NASA93 (NASA data up to 1993):

rank	treatment	median	IQR
1	COCONUT	12	53
1	COCONUT:c0.5,r8	13	46
1	COCOMO-II	14	49
1	COCONUT:c0.25,r8	15	48
2	COCONUT:c1,r8	17	61
2	COCONUT:c1,r4	18	59
2	COCONUT:c0.25,r4	20	44
2	COCONUT:c0.5,r4	21	67

COC81 (original data from the 1981 COCOMO book):

rank	treatment	median	IQR
1	COCOMO-II	3	18
1	COCONUT	4	29
1	COCONUT:c1,r4	5	15
1	COCONUT:c0.5,r4	5	20
2	COCONUT:c0.25,r4	6	30
2	COCONUT:c0.5,r8	7	22
2	COCONUT:c0.25,r8	7	36
2	COCONUT:c1,r8	8	25

Fig. 12 COCOMO vs simpler COCOMO. Displayed as per Fig. 9

- **RQ1**: just using LOC for estimation is far worse that parametric estimation over many attributes (see Section 4.1);
- **RQ2**: new innovations in effort estimation have not superseded parametric estimation (see Section 4.2);
- **RQ3**: Old parametric tunings are not out-dated (see Section 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 Section 4.3);

Hence, we conclude that in 2016, it is still a valid and a recommended practice to *first* try parametric estimation. In these experiments, four to eight projects were enough to learn good predictors (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.

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 Fig. 5, and then processing that data using COCOMO-II.

7 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 appropriate 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 multi-core 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 (Ingold et al. 2013). Finally, the emerging community-based software development, i.e. software crowd sourcing (Li et al. 2013), 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.

Acknowledgments 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

Arcuri A, Briand L (2011) A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: ICSE'11, pp 1–10

Auer M, Trendowicz A, Graser B, Haunschmid E, Stefan B (2006) Optimal project feature weights in analogy-based cost estimation: improvement and limitations. IEEE Trans Softw Eng 32:83–92

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

Black R, Curnow R, Katz R, Bray M (1977) Bcs software production data, final technical report radc-tr-77-116. Technical report Boeing Computer Services, Inc

Boehm B (1981) Software engineering economics. Prentice Hall, Englewood Cliffs

Boehm B (2000) Safe, simple software cost analysis. IEEE Softw:14–17

Boehm B, Horowitz E, Madachy R, Reifer D, Bradford KC, Steece B, Winsor Brown A, Chulani S, Abts C (2000) Software Cost Estimation with Cocomo II. Prentice Hall, Englewood Cliffs

Breiman L, Friedman JH, Olshen RA, Stone CJ (1984) Classification and regression trees

Burgess CJ, Lefley Martin (2001) Can genetic programming improve software effort estimation? a comparative evaluation. Inf Softw Technol 43(14):863–873

Chen Z, Boehm B, Menzies T, Port D (2005) Finding the right data for software cost modeling. IEEE Softw 22:38–46

Chen Z, Menzies T, Port D (2005) Feature subset selection can improve software cost estimation. In: PROMISE'05. Available from http://menzies.us/pdf/05/fsscocomo.pdf

Chulani S, Boehm B, Steece B (1999) Bayesian analysis of empirical software engineering cost models. IEEE Trans Softw Eng 25(4)

Cohen PR (1995) Empirical methods for artificial intelligence, MIT Press, Cambridge

Corazza A, Di Martino S, Ferrucci F, Gravino C, Sarro F, Mendes E (2010) 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, pp 4:1–4:10

Cordero R, Costamagna M, Paschetta E (1997) A genetic algorithm approach for the calibration of cocomo-like models. In: 12th COCOMO Forum

Dabney JB (2002) Return on investment for IV&V. NASA funded study. Results Available from http://sarpresults.ivv.nasa.gov/ViewResearch/24.jsp

Dejaeger K, Verbeke W, Martens D, Baesens B (2012) Data mining techniques for software effort estimation: a comparative study. IEEE Trans Softw Eng 38:375–397

Efron B, Tibshirani RJ (1993) An introduction to the bootstrap. Mono. Stat. Appl. Probab. Chapman and Hall, London

Freiman F, Park R (1979) Price software model - version 3: An overview. In: Proceedings IEEE-PINY workshop on quantitative software models, IEEE catalog number TH 0067-9, pp 32–41

Herd J, Postak J, Russell W, Stewart J (1977) Software cost estimation study-study results, final technical report, radc-tr-77-220. Technical report, Doty Associates

Ingold D, Boehm B, Koolmanojwong S (2013) A model for estimating agile project process and schedule acceleration. In: ICSSP 2013, pp 29–35

Jensen R (1983) An improved macrolevel software development resource estimation model, pp 88–92

Jorgensen M (2015) The world is skewed: ignorance, use, misuse, misunderstandings, and how to improve uncertainty analyses in software development projects, 2015 CREST workshop. http://goo.gl/0wFHLZ

Jørgensen M, Gruschke TM (2009) The impact of lessons-learned sessions on effort estimation and uncertainty assessments. IEEE Trans Softw Eng 35(3):368–383

Jørgensen M, Shepperd M (2007) A systematic review of software development cost estimation studies. Available from http://www.simula.no/departments/engineering/publications/Jorgensen.2005.12

Jorgensen M (2004) A review of studies on expert estimation of software development effort. J Syst Softw 70(1-2):37–60

Li M, Mao K, Yang Y, Harman M (2013) Pricing crowdsourcing-based software development tasks. In: ICSE, new ideas and emerging results, San Francisco, CA, USA, pp 1205–1208

Kadoda G, Cartwright M, Chen L, Shepperd M (2000) Experiences using casebased reasoning to predict software project effort

Kampenes VigdisBy, Dybå T, Hannay JE, Sjøberg DIK (2007) A systematic review of effect size in software engineering experiments. Inf Softw Technol 49(11–12):1073–1086

Keung JW (2008) Empirical evaluation of analogy-x for software cost estimation. In: ESEM '08: international symposium on empirical software engineering and measurement. ACM, New York, NY, USA, pp 294–296

Keung JW, Kitchenham B (2008) Experiments with analogy-x for software cost estimation. In: ASWEC '08: proceedings of the 19th Australian conference on software engineering. IEEE Computer Society, Washington, DC, USA, pp 229–238

Keung JW, Kitchenham BA, Jeffery DR (2008) Analogy-x: providing statistical inference to analogy-based software cost estimation. IEEE Trans Softw Eng 34(4):471–484

Kirsopp C, Shepperd M (2002) Making inferences with small numbers of training sets. IEEE Proc:149

Kocaguneli E, Menzies T, Bener A, Keung J (2012) Exploiting the essential assumptions of analogy-based effort estimation. IEEE Trans Softw Eng 28:425–438. Available from http://menzies.us/pdf/11teak.pdf

Kocaguneli E, Menzies T, Keung JW (2012) On the value of ensemble effort estimation. IEEE Trans Softw Eng 38(6):1403–1416

Kocaguneli E, Menzies T, Keung J, Cok D, Madachy R (2013) Active learning and effort estimation: finding the essential content of software effort estimation data. IEEE Trans Softw Eng 39(8):1040–1053

Kocaguneli E, Menzies T, Mendes E (2014) Transfer learning in effort estimation. Empir Softw Eng:1–31

Kocaguneli E, Zimmermann T, Bird C, Nagappan N, Menzies T (2013) Distributed development considered harmful? In: ICSE, pp 882–890

Li Jingzhou, Ruhe Guenther (2006) A comparative study of attribute weighting heuristics for effort estimation by analogy. In: International symposium on empirical software engineering, p 74

Li J, Ruhe G (2007) Decision support analysis for software effort estimation by analogy. In: PROMISE '07: proceedings of the third international workshop on predictor models in software engineering, p 6

Li J, Ruhe G (2008) Analysis of attribute weighting heuristics for analogy-based software effort estimation method aqua+. Empir Softw Eng 13:63–96

Li Y, Xie M, Goh T (2009) A study of the non-linear adjustment for analogy based software cost estimation. Empir Softw Eng:603–643

Lokan C, Mendes E (2006) 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

Lokan C, Mendes E (2009) Applying moving windows to software effort estimation. In: 3rd international symposium on empirical software engineering and measurement, 2009. ESEM 2009, pp 111–122

Menzies T, Butcher A, Cok DR, Marcus A, Layman L, Shull F, Turhan B, Zimmermann T (2013) Local versus global lessons for defect prediction and effort estimation. IEEE Trans Softw Eng 39(6):822–834. Available from http://menzies.us/pdf/12localb.pdf

Menzies T, Chen Z, Hihn J, Lum K (2006) Selecting best practices for effort estimation. IEEE Trans Softw Eng. Available from http://menzies.us/pdf/06coseekmo.pdf

Menzies T, Dekhtyar A, Distefano J, Greenwald J (2007) Problems with precision. IEEE Trans Softw Eng. http://menzies.us/pdf/07precision.pdf

Menzies T, Kocagüneli E, Minku L, Peters F, Turhan B (2015) Chapter 20 - ensembles of learning machines. In: Sharing data and models in software engineering, pp 239–265

Menzies T, Peters F, Marcus A (2013) Ooops... (errata report for "Better Cross-Company Learning"). In: MSR'13. http://www.slideshare.net/timmenzies/msr13-mistake

Menzies T, Port D, Chen Z, Hihn J, Stukes S (2005) Validation methods for calibrating software effort models. In: Proceedings, ICSE. Available from http://menzies.us/pdf/04coconut.pdf

Menzies T, Shepperd M (2012) Special issue on repeatable results in software engineering prediction. Empir Softw Eng 17(1–2):1–17

Miller A (2002) Subset selection in regression, 2nd edn. Chapman & Hall, London

Minku LL, Yao X (2011) A principled evaluation of ensembles of learning machines for software effort estimation, vol 106

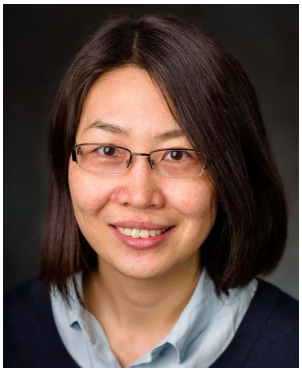
Minku LL, Yao X (2013) Ensembles and locality: insight on improving software effort estimation. Inf Softw Technol 55:1512–1528

Minku LL, Yao X (2014) How to make best use of cross-company data in software effort estimation? In: ICSE'14, pp 446–456

Mittas N, Angelis L (2013) Ranking and clustering software cost estimation models through a multiple comparisons algorithm. IEEE Trans Softw Eng 39(4):537–551

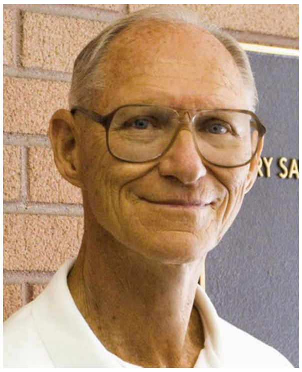
Molokken-Pstvold K, Haugen NC, Benestad HC (2008) Using planning poker for combining expert estimates in software projects. J Syst Softw 81:2106–2117
Murphy-Hill E, Parnin C, Black AP (2012) How we refactor, and how we know it. IEEE Trans Softw Eng 38(1):5–18
Myrtveit I, Stensrud E, Shepperd M (2005) Reliability and validity in comparative studies of software prediction models. IEEE Trans Softw Eng 31(5):380–391
Papakroni V (2013) Data carving: identifying and removing irrelevancies in the data. Master's thesis, Lane Department of Computer Science and Electrical Engineering, West Virginia University
Park R (1988) The central equations of the price software cost model. In: 4th COCOMO users group meeting
Passos C, Braun AP, Cruzes DS, Mendonca M (2011) Analyzing the impact of beliefs in software project practices. In: ESEM'11
Popper KR (1963) Conjectures and refutations. Routledge and Kegan Paul
Posnett D, Filkov V, Devanbu P (2011) Ecological inference in empirical software engineering. In: Proceedings of ASE'11
Putnam L (1976) A macro-estimating methodology for software development, pp 38–43
Scanniello G, Gravino C, Marcus A, Menzies T (2013) Class level fault prediction using software clustering. In: IEEE/ACM 28th international conference on automated software engineering, (ASE), 2013. IEEE, pp 640–645
Shaw M (2001) The coming-of-age of software architecture research. In: Proceedings of the 23rd international conference on software engineering, ICSE '01, vol 656. IEEE Computer Society, Washington, DC, USA
Shepperd M, Schofield C (1997) Estimating software project effort using analogies. IEEE Trans Softw Eng 23(12). Available from http://www.utdallas.edu/~rbanker/SE_XII.pdf
Shepperd MJ, Macdonell SG (2012) Evaluating prediction systems in software project estimation. Inf Softw Technol 54(8):820–827
Spareref.com (2002) Nasa to shut down checkout & launch control system. http://www.spaceref.com/news/viewnews.html?id=475
Stanley C, Byrne MD (2013) Predicting tags for stackoverflow posts. In: Proceedings of ICCM, vol 2013
Valerdi R (2011) Convergence of expert opinion via the wideband delphi method: an application in cost estimation models. In: Incose International Symposium, Denver, USA. Available from http://goo.gl/Zo9HT
Walkerden Fiona, Jeffery R (1999) An empirical study of analogy-based software effort estimation. Empir Softw Engg 4(2):135–158
Walston C, Felix C (1977) A method of programming measurement and estimation. IBM Syst J 16(1):54–77
Whigham PA, Owen CA, Macdonell SG (2015) A baseline model for software effort estimation. ACM Trans Softw Eng Methodol 24(3):20:1–20:11
Wolverton R (1974) The cost of developing large-scale software. IEEE Trans Comput:615–636

Dr. Tim Menzies (Ph.D., UNSW, 1995) is a full Professor in CS at North Carolina State University where he teaches software engineering, automated software engineering, and foundations of software science. He is the author of over 250 referred publications and editor of three recent books summarized the state of the art in software analytics. He has been a lead researcher on projects for NSF, NIJ, DoD, NASA, USDA, as well as joint research with private companies. He is an associate editor of IEEE Transactions on Software Engineering, Empirical Software Engineering, the Automated Software Engineering Journal the Big Data Journal, Information Software Technology, and the Software Quality Journal. For more details, see his home page http://menzies.us.



Dr. Ye Yang is an Associate Professor in the School of Systems and Enterprises. Her research lies in the area of empirical software engineering including software cost estimation, defect prediction, and software process modeling method.

Mr. George Mathew is a PhD candidate in Computer Science from North Carolina State University(NCSU). He received his MS in Computer Science from NCSU in 2016 and B.Tech in Electronics and Instrumentation Engineering from Amrita University in India in 2014. He is part of the RAISE research group in NCSU. His research interests lies in Software Effort Estimation, Optimizing Requirements Engineering Models, Distributed Multi Objective Optimization and Topic Modeling on Graph Data.



Dr. Barry Boehm is a USC Distinguished Professor and the TRW Professor in the USC Computer Sciences, Industrial and Systems Engineering, and Astronautics Departments. He is also the Chief Scientist of the DoD-Stevens- USC Systems Engineering Research Center, and the founding Director of the USC Center for Systems and Software Engineering. He was director of DARPA-ISTO 1989-92, at TRW 1973- 89, at Rand Corporation 1959-73, and at General Dynamics 1955-59. His contributions include the COCOMO family of cost models, the Spiral family of process models, and the Theory W (win-win) approach for creating and evolving successful systems. He is a Fellow of the primary professional societies in computing (ACM), aerospace (AIAA), electronics (IEEE), systems engineering (INCOSE), and Lean methods (LSS), and a member of the U.S. National Academy of Engineering.

Dr. Jairus Hihn (Ph.D, U. Maryland) is a Principal Member of the Systems Engineering staff at the Jet Propulsion Laboratory. He has been active in infusing the use of quantitative management and planning methods across NASA especially in the area of cost estimation. He is the Risk and Programmatics chair on JPLs concurrent engineering team, Team X. He has over 60 publications including recent articles in IEEE Transactions in Software Engineering, Automation in Software Engineering, and Innovations in Systems and Software Engineering. Jairus received ISPAs Parametrician of the Year Award and the USC SSE Lifetime Achievement.