Impacts of Bad ESP (Early Size Predictions) on Software Effort Estimation

George Mathew[a,*], Tim Menzies[a], Jairus Hihn[b]

[a]*Department of Computer Science, North Carolina State University, Raleigh, NC, USA*
[b]*Jet Propulsion Laboratory, Pasadena, CA, USA*

Abstract

Context: Early size predictions (ESP) can lead to errors in effort predictions for software projects. This problem is particular acute in parametric effort models that give extra weight to size factors (for example, the COCOMO model assumes that effort is exponentially proportional to project size).
Objective: Are effort estimates crippled by bad ESP?
Method: Document inaccuracies in early size estimates. Using those error sizes to determine the implications of those inaccuracies via an Monte Carlo perturbation analysis of effort models and an analysis of the equations used in those effort models.
Results: While many projects have errors in ESP of up to $\pm$ 100%, those errors add very little to the overall effort estimate error. Specifically, we find no statistically significant difference in the estimation errors seen after increasing ESP errors from 0 to $\pm$ 100%. An analysis of effort estimation models explains why this is so: the net additional impact of ESP error is relatively small compared to the other sources of error associated within estimation models.
Conclusion: ESP errors effect effort estimates by a relatively minor amount. As soon as a model uses a size estimate *and other factors* to predict project effort, then ESP errors are not crippling to the process of estimation.

Keywords: software effort, parametric models, COCOMO.

1. Introduction

Poor software effort estimation can cripple a project. In the worst case, over-running projects are canceled and the entire development effort is wasted. One challenge with generating software effort estimates is *Early Size Prediction*. ESP is the process of generating a size estimate, early in the project life cycle. For large projects, or projects with extensive government or regulatory oversight, such estimates are needed for:

- When justifying cost for different design decisions;
- When debating which projects get funded or not;
- After the project is delivered late(or not at all) and the management's initial decisions are being audited.

Effort estimation models use a variety of project factors, including the early size prediction to predict development effort. Size estimates can be expressed in several forms including function points [1] or thousands of lines of source code (KLOC) [2]. Incorrect size estimates can have a large impact on effort estimates. Boehm [2] proposes a parametric model of effort estimation that assumes size was proportional exponentially to effort; i.e.

$$effort = X * KLOC^{Y} \quad (1)$$

where X, Y represent numerous "context variables" such as analyst experience. Equation 1 is the basis for the COCOMO model [2, 3] (described in detail later in this paper). In this equation, ESP errors in KLOC can lead to exponentially large errors in the effort estimate. This is a concern since it may be very difficult to estimate KLOC early in the software development process. For example, here is a list of issues that needs to be resolved in order to make a highly accurate size estimate:

*Corresponding author: Tel:1-614-535-8678(George)
Email addresses: george.meg91@gmail.com (George Mathew), tim.menzies@gmail.com (Tim Menzies), jairus.m.hihn@jpl.nasa.gov (Jairus Hihn)

1. How was reused code counted in the size estimate?
2. Was size measured from end-statement or end–line?
3. How were lines of comments handled?
4. How to estimate the size of systems written in multiple languages?
5. How to guess early in the life cycle how much subsequent feedback will change the goals of the project and the size of the delivered system?

This paper argues that while these five points are *potential* problems, the actual net effect is relatively minor. What we will show is that software effort models make estimates based on a large number of parameters, only one of which is KLOC. Hence, the impact of bad ESP may be *relatively less important* than other factors. To defend this claim, this paper explores three research questions:

RQ1: How big are real world ESP errors?

Using data from dozens of projects, we find that:

Result 1
Many real-world software projects usually have ESP errors of up to ± 100%.

RQ2: What is the impact of real-world ESP errors?

A Monte Carlo perturbation analysis was used to understand the impact of the ESP errors found in **RQ1**. For the 265 real-world data sets shown in Figure 1, 100 times, the size estimate of some project was perturbed by 0 to ± 100%. The development effort for project, with the perturbed size, was then calculated. Those estimate where compared to the actual effort (in the non-perturbed data). Given the exponential form of Equation 1, it was expected that this would lead to an exponentially larger effort estimation error. Surprisingly, in our 265 projects. this turned out not to be the case:

Result 2
In 265 real-world projects, ESP errors of up to ± 100% lead to estimate errors of only ± 25%.

This is surprising: in many real-world projects, large ESP errors lead to small estimation errors. To explain this effect, we turned to our next research question.

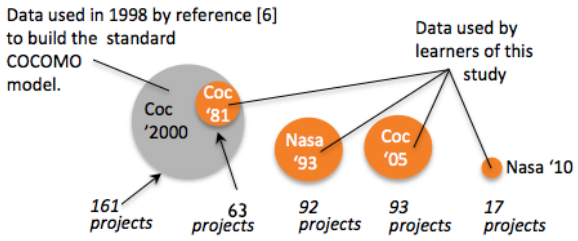

Figure 1: The 265 projects used in **RQ2** come from four data sets. Note that NASA'93 and COC'05 and Nasa'10 have no overlap with the data used to define the version of COCOMO used in this paper.

RQ3: Within an effort estimation model, what is the maximum effect of making large changes to a size estimate?

While **RQ2** explored 265 real-world data sets, **RQ3** explored thousands of synthetic projects. Within a parametric effort estimation model, we moved all the project description attributes across a range the minimum to maximum value. At random points in those ranges, effort estimates were generated using the model. In this study, it was observed:

Result 3
In simulations over thousands of software projects, as KLOC increases, the resulting effort estimates increases much less *than an exponential rate.*

We show via an analytical study that this result can be explained with respect to internal structure of the parametric model used in **RQ3**. Using some algebraic manipulations of our effort estimation model, we can derive expressions from the (a) minimum and (b) maximum possible effort estimate of this model. By dividing these two expressions, it is possible to create a fraction showing the relative effect of changing size estimates, or any other estimates, within this model. In that fraction:

Result 4
The net additional impact of ESP error is relatively small compared to the other sources of error associated within estimation models.

Hence, our conclusion is that ESP errors can degrade project effort estimates (see **RQ2**). However, the size of

that effect is much less than commonly feared. Accordingly, we conclude that ESP errors *are not* the dominate factor leading to inaccurate effort estimates (see **RQ3**).

The rest of this paper offers some background notes on effort estimation. It then presents the data, the methods, and experimental methods used to explore the above questions. After that, the above results will be presented and discussed in detail.

Before begining, it is important to stress that the goal of this paper is *not* to propose a better method for effort estimation. Rather, our goal is to is to say that an issue at the heart of all estimation (guessing the properties of something before that something is built) is not a major problem. This has several implications:

- *Practical implications #1:* This paper removes a major objection to the use of effort estimation models.

- *Practical implications #2:* Dozens of the projects studied in this paper come from very speculative systems (NASA flight systems) or incremental systems where it is most likely that ESP will be inaccurate. Yet this paper shows that the net effect of to inaccuracies is very small. Accordingly, we say that the results of this paper demonstrate that software engineers could make more use of effort estimation even when exploring incremental or highly experimental methods.

- *Theoretical implications:* This paper offers a methodology for testing the impact of size inaccuracies on effort estimation models. In terms of future publications that cite this work, we anticipate that that methodology will be the most widely used part of this paper.

- *Methodological implications:* Numerous recent publications caution that merely because some belief that are widely held by supposed experts, they can still be misleading [4, 5, 6, 7, 8, 9, 10, 11, 12]. All the evidence required to make the analysis of this paper has been available since 2010– yet in all that time no one has thought to use it to test the SE truism that ESP can be very problematic. We hope that this paper inspires other researchers to revisit old truisms in this field (which may not be outdated).

2. Background

2.1. On the Importance of Size Estimates

Several other authors have discussed issues related to size estimates in effort estimation. One class of comment is that measuring size is a meaningless measure and we just should not do it. Quoting the former CEO of Microsoft, Bill Gates [13]:

> *"Measuring software productivity by lines of code is like measuring progress on an airplane by how much it weighs."*

A similar complaint is made by Dijkstra [14] who says

> *"This (referring to measuring productivity through KLOC) is a very costly measuring unit because it encourages the writing of insipid code, but today I am less interested in how foolish a unit it is from even a pure business point of view."*

On the other hand, several studies report that size estimates have a place in effort estimation:

- Walkerden and Jeffery comment that estimating effort via linear size adjustment to an analogue is more accurate than estimates based on regression model [15].

- Kirsopp et al.agree with Walkerden and Jeffery: in a follow-up paper they noted that linear size scaling adaptation results in statistically significant improvements in predicting effort [16].

- Jørgensen et al. observe several industrial software development and maintenance projects. They note that that the effort estimates provided by software professionals (i.e. expert estimates) are to some extent are based on adjustments to size by regressing towards the mean(RTM). [17]. Which is to say that the predictions of human estimators are also Thus, an error in estimating the size of the project can lead to drastic change in predicting the effort of the project.

This paper offers a middle ground between those who claim size estimates are irrelevant and those who say they are fundamental to the estimation process. Like Walkerden, Jeffery, Kirsopp, and Jørgensen et al., we say that size estimates matter somewhat (see the **RQ2** results).

However, as suggested by Gates and Dijkstra, size estimates are not vitally important to effort estimation since effort error comes from two factors:

- Uncertainty in the size estimate;
- Uncertainty in all the other project factors used in the estimation models;

and our **RQ3** results show that uncertainties in the size estimates effect the effort estimate much less than uncertainties in the other project factors.

2.2. Models of Effort Estimation

There are many models of effort estimation. For example, Shepperd et al. prefer non-parametric analogy-based estimation (ABE) methods [18]. Meanwhile, within the United States Department of Defense and NASA, parametric effort models like COCOMO [2] are used extensively and have found to be quite effective [19]. Also, advocates of agile and devops methods prefer the planning poker method (discussed in §2.4). Finally Jørgensen et al. [4] prefer expert-based approaches where estimates are derived by committees of human experts. For studies comparing these methods, or how to usefully combine them, and how to reduce errors when they are used, see [20, 21, 22, 23].

As to the effect of bad ESP on human-generated estimates, we repeat the comments of Jørgensen and Gruschke [4]: the best way to reduce error in human-generated estimates is to not fix bad ESP but to conduct lessons-learned sessions after each project.

As to the effects of bad ESP on non-parametric models, it is trivially simple to show that bad ESP only has minimal effect on ABE. ABE generates estimates via a distance metric defined over all F factors that typically includes one size estimate[1]. ABE does not give extra special weight to the size factor. On the contrary, ABE often mutes the impact of the size factor. The range of values in a size factor may be much larger than all the other factors; e.g. programmer capability may be scored on a six point integer scale while the estimates may range from zero to millions of lines of code. When faced with scales of very different sizes, standard practice [20, 24] is to normalize all the numerics min to max, zero to one using $(x - min)/(max - min)$.

Note that such normalization is recommended practice. Kocaguneli et al. [25], studied the effects of normalization within 90 varieties of effort estimation. That study found 13 "superior" methods and 77 "inferior" ones. Of the eight varieties that never normalized, seven were found in the "inferior" set.

The point here is that once the size factor is muted via normalization, then ABE estimates are effected by bad ESP by a factor of $1/F$. Typically values for F are 5 to 24 with medians of around ten. That is, when compared to the other $1 - 1/F$ factors, bad ESP has relatively little impact on the ABE effort estimate.

A similar argument can be made for other non-parametric methods such as ensemble methods [25] or random forests [26] of CART regression trees [27]. When those models contain F factors only one of which is a size estimate, and those models make no special use of size, then here exists $1 - 1/F$ other factors that contribute more to effort estimation error.

The problem of bad ESP is most acute for parametric models such as Equation 1 that give extra special weight to the size estimates. Such parametric models are widely used, particularly for large government project. In our work with the Chinese and the United States software industry, we see an almost exclusive use of parametric estimation tools such as those offered by Price Systems (pricesystems.com) and Galorath (galorath.com). Also, 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[2].

This paper uses the COCOMO model as a representative of the parametric models since it is open source. As to other parametric effort models, Madachy and Boehm [28] report that many aspects of this model are shared by other models in widespread commercial use such as SEER-SEM [3] and Price-S (now called True S).

[1] Evidence: in the 13 effort data sets of the PROMISE repository http://openscience.us/repo/effort, only one data set has more than a single size attribute.

[2] See the web sites `http://tiny.cc/iceaa`, `http://tiny.cc/nasa_cost`, `http://tiny.cc/csse`

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

Figure 2: COCOMO-II attributes.

2.3. COCOMO Details

COCOMO was developed in two states: an initial release in 1981 [2] followed by an extensive revision in 2000 [3]. In between those releases, Boehm created a consortium for industrial users of COCOMO. This consortium collected information on 161 projects from commercial, aerospace, government, and non-profit organizations. Using that new data, in 2000, Boehm and his colleagues developed a set of *tunings* for COCOMO-II that mapped the project descriptors (very low, low, etc) into the specific attributes used in the COCOMO model (see Figure 2). Those tunings, mappings, and attributes became the COCOMO-II model released

$$effort = a\prod_i EM_i * KLOC^{b+0.01\sum_j SF_j} \quad (2)$$

Here, *EM,SF* are effort multipliers and scale factors and a,b are the *local calibration* parameters (with default values of 2.94 and 0.91). In COCOMO-II, effort multipliers change effort by a linear amount while scale factors change effort by an exponential amount. COCOMO-II reports *effort* as "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.

For a complete implementation of the COCOMO-II effort model, see Figure 3. Note that Equation 2 defines the internal details of X,Y terms in Equation 1: $X = a\prod_i EM_i$ and $Y = b + 0.01\sum_j SF_j$.

```
_  = 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)
```

Figure 3: 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 2.

2.4. Alternatives to COCOMO

Note to reviewers: we are not sure if this paper requires this section. None of the material here is needed for the rest of the paper. Your comments on this matter would be appreciated.

COCOMO was initially designed in the age of waterfall development where projects developed from requirements to delivery with very little operational feedback. Hence, a frequently asked question about this work is the relevancy of that 20^{th} century software management tool to current practices. At issue here is the core question addressed by this paper. While there is nothing inherently "waterfall" within the COCOMO equations, COCOMO does assume that a size estimate is available before the work starts. Hence, it is important to understand when changes to the size of the software results in inaccurate COCOMO estimates.

Another complaint against the COCOMO equations is that such "model-based" methods are less acceptable to humans than "expert-based methods" were estimated are generated via committees of experts. The advantage of such expert-based methods is that if some new project has some important feature that is not included in the COCOMO equations, then human expertise can be used to incorporate that feature into the estimate. However, such expert-based approaches have their limitations. Valerdi [29] lists the cognitive biases that can make an expert offer poor expert-based estimates. Passos et al. offer specific examples for those biases: they show that many commercial software engineers generalize from their first few projects for all future projects [7]. Jørgensen & Gruschke [4] offer other results consistent with Passos et al. when they document how commercial "gurus" rarely use lessons from past projects to improve their future expert-estimates. More generally, Jørgensen [30] reviews studies comparing model- and expert- based estimation and concludes that there there is no clear case that expert-methods are better. Finally, in 2015, Jørgensen further argued [31] 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.

One expert-based estimation method preferred by advocates of agile/devops is "planning poker" [32] 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. Despite having numerous advocates, there are very few comparative studies of planning poker vs parametric methods. The only direct comparison we can find is the 2015 Garg and Gupta study that reports planning poker's estimates can be twice as bad as those generated via parametric methods [22]. One reason for the lack of comparisons is that COCOMO and planning poker usually address different problems:

- COCOMO is often used to negotiating resources prior to starting a project;
- Planning power is often used to adjust current activity

Complexity	N	Product Line	Environment	State of the Art
Simple	2	Existing	Existing	Current
Routine	10	New	Existing	Current
Moderate	14	New	New	Current
Difficult	24	New	New	New

N	Application types
31	Command and Control
6	Office Automation, Software Tools, Signals
5	DFs, Diagnostic, Mission Plans, Sims, Utils;
5	Testing;
3	Operating System

N	Development processes
15	Waterfall
12	Incremental
8	Spiral;
15	Undefined

Figure 4: 50 projects from [33].

within the resource allocation of a project.

Hence we say there is no dispute between planning poker (that is an intra-project task adjustment tool) and COCOMO (which is an pre-project tool for checking if enough resources are available to start a project).

3. Answers to Research Questions

3.1. RQ1: How big are real world ESP errors?

In order to find real-world ESP errors, we look to the historical record. After much investigation, we found two sources that mentioned ESP errors:

- Source #1 covers fifty projects documented by Jones & Hardin [33] from the U.S. Department of Defense.
- Source #2 covers 14 projects from NASA.

Figure 4 describes the projects in Source #1. While the majority of these were military-specific applications (command and control functions), 17 of these are applications types that could be seen in non-military domains. Only a minority of these ($\frac{15}{50} = 30\%$) projects were waterfall projects where requirements were mostly frozen prior to coding; For the remaining 35 of the Jones & Hardin projects, there was ample opportunity for scope creep that could lead to inaccuracies in early life cycle estimates.

Figure 5 shows the relationship between early life cycle estimates of KLOC vs final size for the 50 projects of Source #1. The diagonal line on that plot shows where the estimate equals the actual. The dashed lines show the range within which the estimate is $\pm$ 100% of the actual. The key observations from this Source #1 data are:

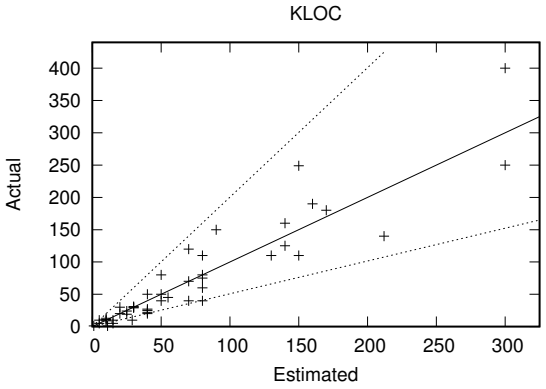

Figure 5: Estimated and actual source lines of code from 50 projects [33].

- The estimates and actuals are often very similar.
- All the estimates fall within $\pm$ 100% of the actual.

In order to check the external validity of these observations from Source #1, we turn to the 14 NASA projects of Source #2. These projects cover the software used in the deep space missions of the Jet Propulsion Laboratory. Due to the highly innovative nature of this kind of software, this final size of this software could easily be incorrected estimated early in its life cycle. Many of the details of those NASA systems is proprietary information so, in this article, we cannot describe the NASA systems at the same level of detail as Source #1. What can be shown, in Figure 6, are the ratios of actual/estimated KLOC values, where the estimates were generated prior to analysis and prior to coding. Usually, these two estimates were similar, but there are exceptions (e.g. the development effort estimate for project M was significantly adjusted after analysis).

The key observation from this Source #2 data is that:

- The $\pm$ 100% error seen in Source #1 covers all but one of the pre-coding NASA estimates.

Hence, our first result is

Result 1
Many real-world software projects usually have ESP errors of up to $\pm$ 100%.

project	pre-analysis	pre-coding
a	-44%	-44%
b	-13%	-4%
c	-6%	-6%
d	-4%	-4%
e	5%	5%
f	7%	
g	10%	10%
h	54%	54%
i	64%	64%
j	69%	69%
k	78%	78%
l	95%	52%
m	206%	10%
n	236%	236%

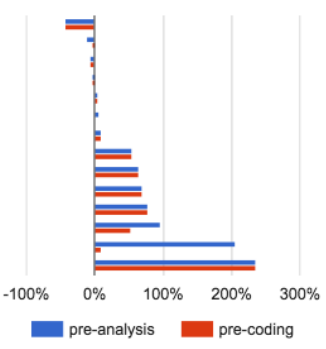

Figure 6: Errors in estimates of final system size (measured in terms of KLOC) seen before analysis and coding in 14 NASA projects. Values in the left-hand-size table are shown graphically at right. All percentages here are percents on the final code size. For example, in the last line, Project N's final size was 236% larger than predicted at the pre-analysis stage. Positive values denote initial under-estimates while negative values denote over-estimates (e.g. the size of the first four projects were initially over-estimated).

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	

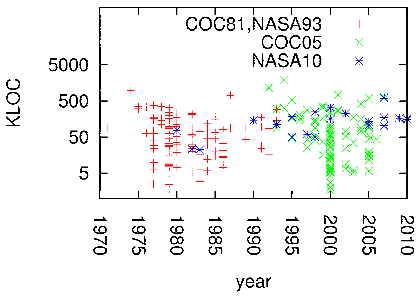

Figure 7: Projects used by the learners in this study. Figure 2 shows project attributes. COC81 is the original data from 1981 COCOMO book [2]. This comes from projects dating 1970 to 1980. NASA93 is NASA data collected in the early 1990s about software that supported the planning activities for the International Space Station. Our other data sets are NASA10 and COC05.

3.2. RQ2: What is the impact of real-world ESP errors?

This section reports the effort errors seen when the size estimate in real world project data are perturbed by up to $\pm$ 100%.

To conduct the perturbation study for identifying a point of tolerance, we use COCOMO since its internal details have been fully published [3]. Also, we can access a full implementation of the 2000 COCOMO model (see Figure 3. Further, we have access to four interesting COCOMO data sets, see Figure 7.

The impact of noise in *KLOC* is studied by varying *KLOC* as follows:

$$KLOC = KLOC * ((1-n) + (2 * n * r)) \quad (3)$$

where $n \in [0.2, 0.4, 0.6, 0.8, 1.0]$ is noise level we are exploring and r is a random number $0 \leq r \leq 1$.

As per the advice of Shepperd and MacDonnell [34], we express effort estimation error as a ratio of some very simple method (in this case, making a prediction by selecting at random some actual effort value from the training data). Shepperd and MacDonnell's argument for this method is as follows: researchers should report their methods as fractions showing how much their method improves over some obvious baseline. Their preferred measure is the SA *standardized error*:

$$SA = \frac{abs(actual - predicted)}{\sum_{i=1}^{1000} |\textbf{\textit{choice}}(all) - predicted|/1000} \quad (4)$$

Figure 8, Figure 9, Figure 10 & Figure 11 show the SA results seen when all examples were passed to COCOMO-II. Since Equation 3 uses a random number generator, we repeated that process 100 times. For all 100 repeated passes through the data, SA was calculated with:

- Estimated project size perturbed as per Equation 3;
- *actual* is the unperturbed value of the effort (taken from the data);
- *predicted* is the estimated value generated using the perturbed size estimate;
- *all* is an array containing all the (not-perturbed) effort values in the dataset;
- ***choice*** is a function that randomly picks one value from an array.

Figure 8: NASA10

Name	Med		IQR	
	Rank	Med	Rank	Med
COCOMO2	1	43	1	1
20%:COCOMO2	1	41	2	13
40%:COCOMO2	1	41	3	28
60%:COCOMO2	2	46	3	34
80%:COCOMO2	2	50	3	44
100%:COCOMO2	2	68	3	49

Figure 9: COC05

Name	Med		IQR	
	Rank	Med	Rank	Med
COCOMO2	1	13	1	1
20%:COCOMO2	1	14	2	8
40%:COCOMO2	1	19	2	14
60%:COCOMO2	2	24	3	25
80%:COCOMO2	2	25	3	25
100%:COCOMO2	2	26	3	30

Figure 10: NASA93

Name	Med		IQR	
	Rank	Med	Rank	Med
COCOMO2	1	14	1	1
20%:COCOMO2	1	14	2	5
40%:COCOMO2	1	15	3	8
60%:COCOMO2	1	16	4	12
80%:COCOMO2	2	20	4	13
100%:COCOMO2	2	27	5	19

Figure 11: COC81

Name	Med		IQR	
	Rank	Med	Rank	Med
COCOMO2	1	3	1	0
20%:COCOMO2	1	4	2	2
40%:COCOMO2	1	4	3	4
60%:COCOMO2	2	6	4	6
80%:COCOMO2	2	6	5	7
100%:COCOMO2	2	8	5	8

In Figure 8, Figure 9, Figure 10 & Figure 11:

- The first column in each table denotes the name of the method using the following nomenclature. "*x%*:COCOMO2" represents COCOMO-II where KLOC is perturbed with an error of *n%* using Equation 3.
- Column 2 shows the "rank" of each result. In those figures, a row's rank increases if its SA results are significantly different than the row above. Note that for error measures like SA, *smaller* ranks are *better*. These ranks were computed using the Scott-Knott test of Figure 3.2. This test was adopted as per the recent recommendations of Mittas and Angelis in IEEE TSE 2013 [35].
- Column 3 shows the median and IQR for the median of the SA over the 100 repeats. The median value of a list is the 50th percentile value while the IQR is the 75th-25th percentile value. Note that for error measures like SA, *smaller* medians and smaller IQRs are *better*.

As might be expected, in these results, as ESP error increases, so to did the SA estimation error:

- For NASA10, from 43 to 68%;
- For COC05, from 13 to 26%;
- For NASA93, from 14 to 27%;
- For COC81, from 3 to 8%.

That said, the size of the increase is surprisingly small. Even with errors up to 100%:

- The increased estimation error was sometimes very small: see the 5% increase in COC81;
- The estimation error was never very large: i.e. never more than the 25% increase seen in COC81.

Moreover, the median of the increased estimation error was usually smaller than the inter-quartile range; e.g. for NASA10, the increase in the median error was 25% while the inter-quartile range for 100% perturbation was 59%. Further, as shown by the "rank" column in these results, all these results were assigned the same value of "rank"=1). That is, while size estimate increases estimation error, those increases *were not statistically significant*.

The reason for this result are clear: the variability associated effort estimates is not small. Figure 8, Figure 9, Figure 10 & Figure 11 report that the inter-quartile range in estimation error can grow as large as 71%. While size estimate error contributes somewhat to that error, it is clear that factors *other than size estimate error* control the estimate error. These other factors are explored further in **RQ3**. Meanwhile, the clear result from **RQ2** is:

Result 2
In 265 real-world projects, ESP errors of up to $\pm$ *100% lead to estimate errors of only* $\pm$ *25%.*

Scott-Knott procedure recommended by Mittas & Angelis in their 2013 IEEE TSE paper [35]. This method sorts a list of l treatments with ls measurements by their median score. It then splits l into sub-lists m,n in order to maximize the expected value of differences in the observed performances before and after divisions. E.g. for lists l,m,n of size ls,ms,ns where $l = m \cup n$:

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

Scott-Knott then applies some statistical hypothesis test H to check if m,n are significantly different. If so, Scott-Knott then recurses on each division.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(endorsed by Arcuri et al. in ICSE '11 [36]) of and non-parametric bootstrap sampling [37]; i.e. our Scott-Knott divided the data if *both* bootstrapping and an effect size test agreed that the division was statistically significant (99% confidence) and not a "small" effect ($A12 \geq 0.6$). For a justification of the use of non-parametric bootstrapping, see Efron & Tibshirani [37, p220-223]. For a justification of the use of effect size tests see Shepperd & MacDonell [34]; Kampenes [38]; and Kocaguneli et al. [39]. 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:

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

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 [36].

Figure 12: Scott-Knott Test

3.3. RQ3: Within an effort estimation model, what is the maximum effect of making large changes to a size estimate?

Recall from the introduction that the exponential nature of the COCOMO equation made it seem as if COCOMO would be most susceptible to errors in lines of code. Yet we saw in the last section that COCOMO is remarkably *insensitive* to KLOC errors. This section checks if that result is just some quirk of the 265 projects studied above, or if it is a more fundamental property.

Equation 1 said $effort = X * KLOC^Y$ where $Y = b + 0.01 \sum_i SF_i$. One explanation for the strange results of the last section is that the Y coefficient on the exponential term are much smaller than the linear X factors; i.e. the

COCOMO effort estimate is effectively linear on X, and not exponential on $KLOC^Y$.

To prove this claim, we examine the coefficients of the terms in the COCOMO equation to see what effect changes in KLOC have on COCOMO's effort prediction The coefficients learned by Boehm in 2000 for the COCOMO were based on an analysis of 161 projects from commercial, aerospace, government, and non-profit organizations [3]. At the time of that analysis, those projects were of size 20 to 2000 KLOC (thousands of lines of code) and took between 100 to 10000 person months to build. Boehm's SF_i coefficients are presented in a table inside the front cover of the COCOMO-II text [40](see Figure 3). When projects have "very low", "low", "nominal", "high", "very high" values in the COCOMO , then from that table it can be see that:

	$0.01 \sum_i SF_i$
very low	0.32
low	0.25
nominal	0.192
high	0.13
very high	0.06

(5)

In 2000, Boehm proposed default values for a,b= $2.94, 0.91$. Those ranges of where checked by Baker [41] using 92 projects from NASA's Jet Propulsion Laboratory. Recall from Equation 2 that the a,b local calibration parameters can be adjusted using local data. Baker checked those ranges by, 30 times, running the COCOMO calibration procedure using 90% of the JPL data (selected at random). He reported that a was approximately linearly related to b as follows:

$$(2.2 \leq a \leq 9.18) \bigwedge (b(a,r) = -0.03a + 1.46 + r * 0.1)$$

Note that Baker's found ranges for a included the $a = 2.94$ value proposed by Bohem.

In the above, "r" is a random number $0 \leq r \leq 1$ so Baker's maximum and minimum b values were:

$$b(2.2,\ 0) = 1.394$$
$$b(9.18,\ 1) = 1.2846$$

Combined with Boehm's default values for $b = 0.91$, we say that in the historical record there is evidence for b ranging

$$0.91 \leq b \leq 1.394 \quad (6)$$

Combining the above with Equation 5, we see that the Y coefficient on the KLOC term in Equation 1 is

	$Y = b + 0.01 \sum_i SF_i$
very low	$1.22 \leq Y \leq 1.71$
low	$1.16 \leq Y \leq 1.65$
nominal	$1.10 \leq Y \leq 1.58$
high	$1.04 \leq Y \leq 1.52$
very high	$0.97 \leq Y \leq 1.46$

(7)

Figure 13 shows $effort = KLOC^{Y}$ results using the coefficients of Equation 7. Note that the vertical axis of that chart a logarithmic scale. On such a scale, an function that is exponential on the horizontal access will appear as a straight line. All these plots bent over to the right; i.e. even under the most pessimist assumptions (see "very low" for "upper bound"). That is:

Result 3
In simulations over thousands of software projects, as KLOC increases, the resulting effort estimates increased much less *than exponentially.*

We show via the following analytical study that **Result 3** can be explained with respect to internal structure of the COCOMO parametric model. Using some algebraic manipulations of our effort estimation model, we can derive expressions from the (a) the minimum and (b) the maximum possible effort estimate from this model. By dividing these two expression, it is possible to create an fraction showing the relative effect of changing size estimates, or any other estimates, within this model.

From Equation 2, 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:

$$0.057 * KLOC^{0.97} \leq effort \leq 115.6 * KLOC^{1.71}$$

The exponents in the this expression come from Equation 7. The linear terms come from the product of the min/max effort multipliers from the COCOMO-II text [3].

Dividing the minimum and maximum values shows how effort can vary for any given KLOC due to variations in the effort multipliers and scale factors:

$$115.6/0.057 * KLOC^{1.71-0.97} = 2028 * KLOC^{0.74} \quad (8)$$

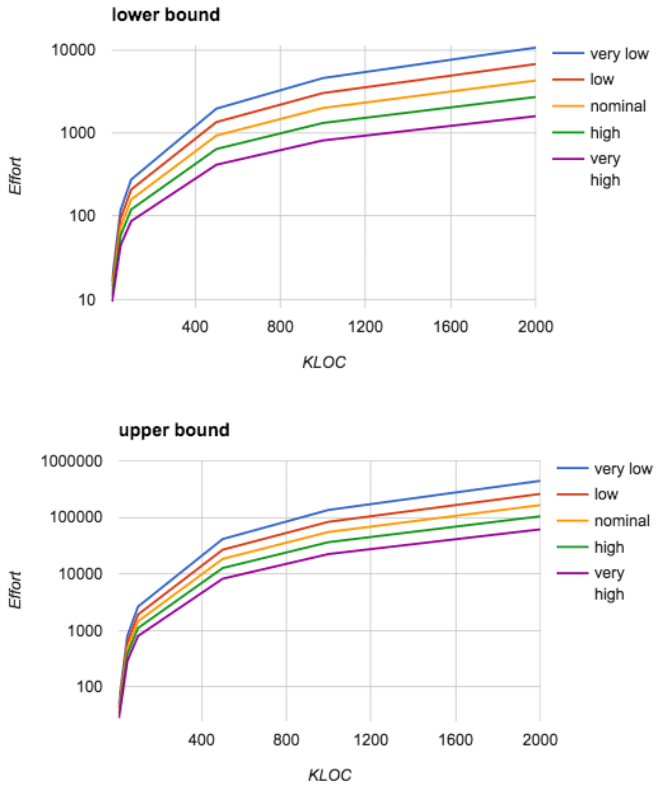

Figure 13: Growth in effort estimates as source code grows. Growth rate determined by Equation 7. "Lower bound" is the most optimistic projection (effort grows slowest as KLOC increases) while "Upper bound is most pessimistic. For example, in 'upper bound" for "very low", $effort = KLOC^{1.71}$ (where 1.71 is the top-right figure of Equation 7).

Note the large linear term (2028) coming from the effort multipliers and the small exponential term (0.74) coming from the scale factors. Equation 8 shows that errors in the effort multipliers can change effort more than errors in the size estimate. Hence:

Result 4
The net additional impact of ESP error is relatively small compared to the other sources of error associated within estimation models.

Note that **Result 3** and **Result 4** explain why the **RQ2** results were not dramatically altered by errors in KLOC.

4. Validity

4.1. Sampling Bias

The perturbation study used in **RQ3** perturbed KLOC values according to the ranges found in 64 projects discussed in **RQ1**. Hence, the **RQ3** conclusions are biases by that sample.

The other studies shown above were based on more data (265 projects). While that 265 does not cover the space of all projects, we note that it is a much larger sample than what is seen in numerous other research papers in this arena.

While our sampling bias is clear, it also shows clearly how to refute our conclusions:

- This study would need to be repeated in the ESP error in real world projects is observed to grow beyond $\pm$ 100%.

4.2. External Validity

One clear bias in this study is the use of the COCOMO model for studying the effects of KLOC errors on estimates. Our case for using COCOMO was made above: (a) COCOMO s widely used in industry and government circles in the United States and China; (b) COCOMO's assumption that effort is exponentially proportional to KLOC seems to make it exponentially sensitive to errors in KLOC estimates; (c) many aspects of COCOMO are shared by other models in widespread commercial use such as SEER-SEM and Price-S (now called True S).

As to external validity of our conclusions for other effort estimation methods, Section 2.2 offered arguments that ESP would have minimal impact on non-parametric models.

5. Discussion

If we accept the external validity of these conclusions, then the next questions are:

- What can be done to reduce the effort errors caused by all the non-size factors?
- Why is software project development so insensitive to ESP errors?

As to the first question, we recommend feature selection. Elsewhere we have shown that as the number of training examples shrink, then it becomes more important to build models using just the few most important factors in the data. Automatic algorithms can find those most important factors [42].

As to the second question, it can be addressed via graph theory. While he never said it, we believe Boehm's core COCOMO assumption (that effort is exponentially proportional to size) comes for the mathematics of communication across networks. In an arbitrarily connected graph, a node has to co-ordinate with up to $2^{(N-1)}$ neighbors. If these nodes are software systems, then each such connection is one more team to co-ordinate with, or one more interaction to check in the test suite. Hence, a linear increase in a system size can lead to exponentially more complex co-ordination and testing.

To avoid this extra exponential effort, it is necessary to reduce the number of other systems that interact with a particular node. This is the task of software architectures [43]. A well-defined software architecture offers clear and succinct interfaces between different parts of a system [44]. Either side of an interface, software may be very complex and inter-connected. However, given an interface that allows limited communication, the number of interactions between different parts of the systems are reduced. We conjecture that these reduced interactions are the core reasons why our models are not reporting effort being overly reactive to bad size predictions.

6. Conclusion

We have offered evidence that we can be optimistic about our ability to generate reasonably accurate early life cycle estimates, despite bad ESP:

- From **RQ1**, we know that ESP errors seen in practice have limited ranges. Looking at Figure 5 and Figure 6 we can see many projects where the estimated size was close to the actual final system.
- In **RQ2**, we perturbed KLOC values within effort predictors (within the maximum ranges found by **RQ1**), The net effect of those perturbations was observed to be very small– in fact statistically insignificant.

- In **RQ3** we checked the generality of the **RQ2** conclusions. When we compared the effects of KLOC error relative to errors in the other project factors, we found that KLOC errors were relatively less influential.

The last point is particularly significant. While there are many reasons why ESP can fail (see the list of 5 points in the introduction), as shown above, the net impact of those errors is relatively small.

In summary, modern effort estimation models use much more than just size predictions. While errors in size predictions increase estimate error, by a little, it is important to consider all the attributes used by effort model. Future work should focus on how to better collect more accurate information about (e.g.) the factors shown in Figure 2.

Acknowledgments

The work has partially funded by a National Science Foundation CISE CCF award #1506586.

References

[1] A. J. Albrecht, J. E. Gaffney, Software function, source lines of code, and development effort prediction: A software science validation, IEEE Trans. Softw. Eng. 9 (6) (1983) 639–648. doi:10.1109/TSE.1983.235271.
URL http://dx.doi.org/10.1109/TSE.1983.235271

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

[3] B. Boehm, E. Horowitz, R. Madachy, D. Reifer, B. K. Clark, B. Steece, A. W. Brown, S. Chulani, C. Abts, Software Cost Estimation with Cocomo II, Prentice Hall, 2000.

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

[5] T. Menzies, Y. Yang, G. Mathew, B. Boehm, J. Hihn, Negative results for software effort estimation, Empirical Software Engineering (2016) 1–26doi:10.1007/s10664-016-9472-2.
URL http://dx.doi.org/10.1007/s10664-016-9472-2

[6] T. Menzies, W. Nichols, F. Shull, L. Layman, Are delayed issues harder to resolve? revisiting cost-to-fix of defects throughout the lifecycle, Empirical Software Engineering (2016) 1–33doi:10.1007/s10664-016-9469-x.
URL http://dx.doi.org/10.1007/s10664-016-9469-x

[7] C. Passos, A. P. Braun, D. S. Cruzes, M. Mendonca, Analyzing the impact of beliefs in software project practices, in: ESEM'11, 2011, pp. 444–452.

[8] P. Devanbu, T. Zimmermann, C. Bird, Belief & evidence in empirical software engineering, in: Proceedings of the 38th International Conference on Software Engineering, ACM, 2016, pp. 108–119.

[9] N. Bettenburg, M. Nagappan, A. E. Hassan, Towards improving statistical modeling of software engineering data: think locally, act globally!, Empirical Software Engineering (2014) 1–42doi:10.1007/s10664-013-9292-6.
URL http://dx.doi.org/10.1007/s10664-013-9292-6

[10] Y. Yang, Z. He, K. Mao, Q. Li, V. Nguyen, B. W. Boehm, R. Valerdi, Analyzing and handling local bias for calibrating parametric cost estimation models, Information & Software Technology 55 (8) (2013) 1496–1511.

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

[12] B. Ray, D. Posnett, V. Filkov, P. Devanbu, A large scale study of programming languages and code quality in github, in: Proceedings of the ACM SIGSOFT 22nd International Symposium on the Foundations of Software Engineering, FSE '14, ACM, 2014, pp. 155–165.

[13] K. Gollapudi, Function points or lines of code? - an insight (2004).

[14] E. W. Dijkstra, On the cruelty of really teaching computing science, http://www.cs.utexas.edu/users/EWD/ewd10xx/EWD1036.PDF, accessed: 2016-11-30 (1988).

[15] F. Walkerden, R. Jeffery, An empirical study of analogy-based software effort estimation, Empirical Softw. Engg. 4 (2) (1999) 135–158. doi:http://dx.doi.org/10.1023/A:1009872202035.

[16] C. Kirsopp, E. Mendes, R. Premraj, M. Shepperd, An empirical analysis of linear adaptation techniques for case-based prediction, in: International Conference on Case-Based Reasoning, Springer, 2003, pp. 231–245.

[17] M. Jørgensen, U. Indahl, D. Sjøberg, Software effort estimation by analogy and regression toward the mean, Journal of Systems and Software 68 (3) (2003) 253–262.

[18] M. Shepperd, C. Schofield, Estimating software project effort using analogies, IEEE Transactions on software engineering 23 (11) (1997) 736–743.

[19] K. Lum, J. Powell, J. Hihn, Validation of spacecraft software cost estimation models for flight and ground systems.

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

[21] L. L. Minku, X. Yao, Ensembles and locality: Insight on improving software effort estimation, Information and Software Technology 55 (8) (2013) 1512–1528.

[22] S. Garg, D. Gupta, Pca based cost estimation model for agile software development projects, in: Industrial Engineering and Operations Management (IEOM), 2015 International Conference on, 2015, pp. 1–7. doi:10.1109/IEOM.2015.7228109.

[23] E. Kocaguneli, T. Menzies, J. Keung, D. Cok, R. Madachy, Active learning and effort estimation: Finding the essential content of software effort estimation data, IEEE Transactions on Software Engineering 39 (8) (2013) 1040–1053.

[24] D. W. Aha, D. Kibler, M. K. Albert, Instance-based learning algorithms, Machine learning 6 (1) (1991) 37–66.

[25] E. Kocaguneli, T. Menzies, J. Keung, On the value of ensemble effort estimation, IEEE Trans. Softw. Eng. 38 (6) (2012) 1403–1416. doi:10.1109/TSE.2011.111.
URL http://dx.doi.org/10.1109/TSE.2011.111

[26] L. Breiman, Random forests, Machine learning 45 (1) (2001) 5–32.

[27] L. Breiman, J. Friedman, C. J. Stone, R. A. Olshen, Classification and regression trees, CRC press, 1984.

[28] R. Madachy, B. Boehm, Comparative analysis of cocomo ii, seer-sem and true-s software cost models, Tech. Rep. USC-CSSE-2008-816, University of Southern California (2008).

[29] R. Valerdi, Convergence of expert opinion via the wideband delphi method, in: 21st Annual International Symposium of the International Council on Systems Engineering, INCOSE 2011, 2011.

[30] M. Jorgensen, A review of studies on expert estimation of software development effort, Journal of Systems and Software 70 (1-2) (2004) 37–60. doi:10.1016/S0164-1212(02)00156-5.

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

[32] K. Molokken-Pstvold, N. C. Haugen, H. C. Benestad, Using planning poker for combining expert estimates in software projects, Journal of Systems and Software 81 (2008) 21062117.

[33] P. H. R. Jones, Software code growth: A new approach based on an analysis of historical actuals, in: DNI Cost Analysis Improvement Group, 2007.

[34] M. J. Shepperd, S. G. MacDonell, Evaluating prediction systems in software project estimation, Information & Software Technology 54 (8) (2012) 820–827.

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

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

[37] B. Efron, R. J. Tibshirani, An introduction to the bootstrap, Mono. Stat. Appl. Probab., Chapman and Hall, London, 1993.

[38] V. B. Kampenes, T. Dybå, J. E. Hannay, D. I. K. Sjøberg, A systematic review of effect size in software engineering experiments, Information & Software Technology 49 (11-12) (2007) 1073–1086.

[39] E. Kocaguneli, T. Zimmermann, C. Bird, N. Nagappan, T. Menzies, Distributed development considered harmful?, in: ICSE, 2013, pp. 882–890.

[40] B. Boehm, Safe and simple software cost analysis, IEEE Software (2000) 14–17.

[41] D. 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, available from https://eidr.wvu.edu/etd/documentdata.eTD?documentid=5443 (2007).

[42] Z. Chen, T. Menzies, D. Port, D. Boehm, Finding the right data for software cost modeling, IEEE Software 22 (6) (2005) 38–46. doi:10.1109/MS.2005.151.

[43] D. Garlan, M. Shaw, An introduction to software architecture, Advances in software engineering and knowledge engineering 1 (3.4).

[44] D. L. Parnas, On the criteria to be used in decomposing systems into modules, Commun. ACM 15 (12) (1972) 1053–1058. doi:10.1145/361598.361623.
URL http://doi.acm.org/10.1145/361598.361623