Accurate Estimates Without Calibration?

Tim Menzies[1], Steve Williams[1], Oussama Elrawas[1], Daniel Baker[1], Barry Boehm[2], Jairus Hihn[3], Karen Lum[3], and Ray Madachy[4] *

[1] LCSEE, West Virginia University, Morgantown, WV, USA, tim@menzies.us, oelrawas@mix.wvu.edu, swilli12@mix.wvu.edu danielryanbaker@gmail.com
[2] CS, University of Southern California, Los Angeles, California, USA, boehm@sunset.usc.edu, madachy@usc.edu
[3] JPL, California, USA, jairus.hihn@jpl.nasa.gov, karen.t.lum@jpl.nasa.gov
[4] SE, Naval Postgraduate School, San Diego CA, USA, rjmadach@nps.edu

Abstract. Most process models calibrate their internal settings using local data. Collecting this data is expensive, tedious, and often an incomplete process. Is it possible to make accurate process decisions without historical data? Variability in model output arises from (a) uncertainty in model inputs and (b) uncertainty in the internal parameters that control the conversion of inputs to outputs. We find that, for USC family process models such as COCOMO and COQUALMO, we can control model outputs by using an AI search engine to adjust the controllable project choices *without* requiring local tuning. For example, in ten case studies, we show that the estimates generated in this manner are very similar to those produced by traditional methods (local calibration). Our conclusion is that, (a) while local tuning is always the preferred option, there exist some process models for which local tuning is optional; and (b) when building a process model, we should design it such that it is possible to use it without tuning.
Word length: 6525 words (4875 words of text + 7 figures at 250 words per figure).

1 Introduction

Process models have many purposes including estimating project parameters or conducting what-if queries to find better ways to organize a project. Standard practice is to calibrate these models using local tuning data. It can be quite difficult to access such data. For example, after 26 years of trying, we have only collected less than 200 sample projects for the COCOMO database. Also, even after two years of effort we were only able to add 7 records to a NASA-wide software cost metrics repository [31].

* Parts of this research as carried out at West Virginia University, the University of Southern California, and the Naval Postgraduate School. Other parts of this research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the 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 United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

There are many reasons for this "data drought" including data not being collected or the business sensitivity associated with the data, as well as differences in how the metrics are defined, collected and archived. For years, we have struggled with the data drought problem and have recommended elaborate feature subset selection methods to prune uninformative data [14, 32]. Here, we take a radically different approach and explore the value of models that have not been tuned.

Process models can be viewed as a set of constraints between inputs and output. In this view, tuning a model using local data *constrains* the model to reduce the *variance* in the model output. It is useful to distinguish two classes of variables, both of which are illustrated in the following simplified COCOMO model:

$$effort = a \cdot LOC^{b+pmat} \cdot acap \tag{1}$$

Here, a, b are *tuning parameters* that control the linear and exponential effects (respectively) on model output. These tuning parameters are adjusted using historical data. On the other hand, $pmat$ (process maturity) and $acap$ (analyst capability) are *project choices* that are adjusted by managers. Project choices are typically model inputs while tuning parameters typically control calculations internal to the model. Both classes of variables introduce variance into the estimates:

- *Tuning variance:* variability in the tuning variables due to training data that is incomplete or noisy;
- *Project variance:* variability in the project choices due to, say, uncertainty about the process maturity of the sub-contractors.

Much research has explored tuning variance reduction (e.g., [8, 11, 18, 24, 26]). In this paper, we take a different approach and ask if estimation variance can be controlled by reducing *only* project variance. Our tool in this study is SEESAW, an AI search engine that conducts large scale what-if queries over software process models. SEEWAW constrains project choices, *but not the tuning variance* in its search for options that reduce effort, defects, threats, and development time estimates. The main result of this paper is that, at least for COCOMO/COQUALMO:

> *The range of estimate errors seen after constraining the project choices (but not the tuning variables) is almost identical to the range seen after constraining just the tuning variables.*

From a business perspective, this result means that certain process models can be used for decision making in one of two ways:

1. *Either* constrain the tuning variance using historical data;
2. *Or* constrain the project choices using an AI search engine like SEESAW.

Note that this second method avoids a lengthy and expensive data collection phase prior to decision making. This result is of tremendous practical benefit since it is often very difficult to find relevant data within a single organization to precisely tune all the internal parameters inside a process model.

Prior reports on our AI search methods [30, 31] were based on limited case studies; here we report ten new case studies showing that our main effect holds in a wide range

of cases. Also, those prior reports failed to check the validity of their results. The ten case studies discussed below show that the distribution of SEESAWS's estimates approximate those seen with conventional methods, despite being generated from a large space of possible tunings. This validity check greatly increases our confidence in the SEESAW method.

The rest of this paper describes how we arrived at our main result. After a discussion of related work, we review the models used in this study and the AI search engine that finds useful constraints to project choices. This is followed by a description of an experiment that compares the range of errors seen after constraining just the project choices (using SEESAW) or just the tuning variables (using linear regression). In our future work, we explore the implications of this work for software process modeling: if there exists a set of process models for which tuning is optional then, where possible, we should favor the usage of such models.

2 Related Work

Equation 1 described a $model$ containing $tuning$ and $project$ parameters. Related work may be categorized according to how they treat $(model, tuning, project)$.

- *Data Mining*: Using different data mining techniques, generate a range of $model$s.
- *Prediction*: fix $model$ and $project$ and generates fixed $estimates$.
- *Calibration*: import an log of $estimates$ and $project$ variables, find changes to $model$ that best explain how $project$ inputs lead to $estimation$ outputs.
- *Monte Carlo* studies: generate estimates from one $model$ varying the $project$ choices.

In the field of effort estimation:

- *Data mining* is useful when the exact best form of the $model$ is unknown [32].
- *Prediction* is used to create one point estimate for a project; e.g., COCOMO [8, 9],PRICE-S [35] and SEER-SEM [21].
- *Calibration* is useful for learning from historical data; e.g., see Boehm's local calibration procedure [8, p526-529] or the COSEEKMO toolkit [32].
- *Monte Carlo* studies are useful for conducting what-if queries across a range of possible projects [39]. Such Monte Carlo studies are conducted by many tools including COBRA [12], Crystal Ball [6], SCAT [27, 28], and 2CEE [7].

To the best of our knowledge, this work is the first to try controlling the $project$ variables while leaving the $tuning$ unconstrained. Even in the field in *search-based software engineering*, we have not seen anything like this study. It is true that search-based SE often uses non-linear search methods like simulated annealing. A recent review of 123 search-based SE papers [41] showed that much of that work relates to testing (e.g., SA to minimize test suites for regression testing) while only a handful of those papers related to the kinds of early project process planning discussed here. For example, Aguilar-Ruiz et.al. [2] and Alvarez et.al. [4] apply search-based methods for effort estimation. One facet that distinguishes SEESAW from other methods is that we are searching over more than just the effort models explored by the Aquilar-Ruiz & Alvarez teams.

Some researchers have explored estimation uncertainty using a Bayesian analysis. For example, Pendharkar et.al. [36] demonstrate the utility of Bayes networks in effort estimation while Fenton and Neil explore Bayes nets and defect prediction [17]. Unlike this paper, neither of these teams links defect models to effort models. We elect to take a non-Bayesian approach since most of the industrial and government contractors we work with use parametric models like COCOMO.

Other researchers use Search-Based Software Engineering (SBSE) optimization techniques and apply techniques taken from operations research and meta-heuristic search (e.g., simulated annealing and genetic algorithms). Typically, SBSE hunts for near optimal solutions to complex and over-constrained software engineering problems. This approach has been applied to many problems in software engineering (e.g., requirements engineering [20]) but most often in the field of software testing [5]. Harmon's writing inspired us try simulated annealing to search the what-ifs in untuned COCOMO models [31]. However, we found that SEESAW ran much faster and produced results with far less variance than simulated annealing.

The process simulation community (e.g., Raffo [38]) studies models far more elaborate than COCOMO or COQUALMO. For example, COCOMO & COQUALMO assume linear parametric equations while other researchers explore other forms:

- discrete-event models [23, 25];
- system dynamics models [1];
- state-based models [3, 19, 29];
- rule-based programs [33];
- standard programming constructs such as those used in Little-JIL [13, 42].

These rich modeling frameworks allow the representation of detailed insights into an organization. However, the effort required to tune them is non-trivial. For example, Raffo spent two years tuning and validation one such model to one particular site [40].

3 Models Used in this Study

3.1 COCOMO, COQUALMO, THREAT

The background to all our work are three USC software process models:

- The COQUALMO software *defect* predictor [10, p254-268]. COQUALMO models two processes (defect introduction and defect removal) for three phases (requirements, design, and coding).
- The COCOMO software *effort* and development *time* predictor [10, p29-57]. COCOMO assumes that effort is exponentially proportional to some *scale factors* and linearly proportional to some *effort multipliers*. COCOMO estimates development months (225 hours) and calendar months and includes all coding, debugging, and management activities.
- The THREAT predictor for *project effort & schedule overrun* [10, p284-291]. The THREAT model contains a large set of two-dimensional tables like Figure 1 representing pairs of variable settings that are problematic. For example, using the $rely$ vs $sced$ table, the THREAT model would raise an alert if our tool decides to build

a system with high $rely$ (required reliability) and low $sced$ (schedule available to the development).

	rely= very low	rely= low	rely= nominal	rely= high	rely= very high
sced= very low	0	0	0	1	2
sced= low	0	0	0	0	1
sced= nominal	0	0	0	0	0
sced= high	0	0	0	0	0
sced= very high	0	0	0	0	0

Fig. 1. An example risk table

From our perspective, these models have several useful features:

- Unlike other models such as PRICE-S [35], SLIM [37], or SEER-SEM [21], the COCOMO family of models is fully described in the literature. Also, at least for the effort model, there exist baseline results [15].
- We work extensively with governemnt agencies writing software. Amongst those agencies, these models are frequently used to generate and justify budgets.
- The space of possible tunings within COCOMO & COQUALMO is well defined (see below). Hence, it is possible to explore the space of possible tunings.
- Even allowing for full variance in the tuning parameters, the estimation variance of COCOMO can be reduced via intelligent selection of input variables. We would consider switching to other models if it could be shown that the variance of these other models could be controlled just as easily.

Figure 2 shows the project choices within COCOMO / COQUALMO / THREAT. The last two columns of this figure show the results of a Delphi panel session at the Jet Propulsion Laboratory where the COCOMO variables were separated into:

- the $tactical$ variables that can be changed within the space of one project;
- the $strategic$ variables that require higher-level institutional change (and so may take longer to change).

For example, the panel declared that $pmat$ (process maturity) is hard to change within the space of a single JPL project. Note that these definitions of $strategic$ and $tactical$ choices are not hard-wired into our system. If a user disagrees with our definitions of strategic/tactical, they can change a simple configuration file.

3.2 Example Projects

Figure 3 summarizes four NASA case studies using the project choices of Figure 2:

- "OSP" is the GNC (guidance, navigation, and control) component of NASA's 1990s *Orbital Space Plane*;

		strategic?	tactical?
scale	prec: have we done this before?	✓	
factors	flex: development flexibility		✓
(exponentially	resl: any risk resolution activities?		✓
decrease	team: team cohesion		✓
effort)	pmat: process maturity	✓	
upper	acap: analyst capability	✓	
(linearly	pcap: programmer capability	✓	
decrease	pcon: programmer continuity	✓	
effort)	aexp: analyst experience	✓	
	pexp: programmer experience	✓	
	ltex: language and tool experience	✓	
	tool: tool use		✓
	site: multiple site development	✓	
	sced: length of schedule		✓
lower	rely: required reliability		
(linearly	data: secondary memory storage requirements		✓
increase	cplx: program complexity		✓
effort)	ruse: software reuse		✓
	docu: documentation requirements		✓
	time: runtime pressure		
	stor: main memory requirements		✓
	pvol: platform volatility		
COQUALMO	auto: automated analysis	✓	✓
defect removal	execTest: execution-based testing tools	✓	✓
methods	peer: peer reviews	✓	✓

Fig. 2. The variables of COCOMO, COQUALMO, and the THREAT model.

- "OSP2" is a later version of OSP;
- "Flight" and "ground" show typical ranges of NASA's Jet Propulsion Laboratory.

Inside our model, project choices typically range from 1 to 5 where "3" is the nominal value that offers no change to the default estimate. Some of the project choices in Figure 3 are known precisely (see all the choices with single *values*). But many of the features in Figure 3 do not have precise values (see all the features that *range* from some *low* to *high* value).

Sometimes the ranges of choices are very narrow (e.g., the process maturity of JPL ground software is between 2 and 3), and sometimes the ranges are very broad. Figure 3 does not mention all the features listed in Figure 2 inputs. For example, our defect predictor has inputs for use of *automated analysis*, *peer reviews*, and *execution-based testing tools*. During SEESAW's search, for all project choices not mentioned in Figure 3, values are picked at random from the full range of Figure 2.

3.3 Defining the Space of Possible Tunings

Many of our project choices have a linear relationship to the output. Such linear relations form the line $y = mx + b$ with slope "m" passing through point $x = 3, y = 1$; i.e., at the nominal value of "3", there are no changes to the effort estimate. Such a line has a y-intercept of $b = 1 - 3m$. Substituting this value of b into $y = mx + b$ yields:

$$estimate = m(x - 3) + 1$$

	float			fixed	
project	variable	low	high	variable	setting
OSP	prec	1	2	data	3
	flex	2	5	pvol	2
	resl	1	3	rely	5
	team	2	3	pcap	3
	pmat	1	4	plex	3
	stor	3	5	site	3
	ruse	2	4		
	docu	2	4		
	acap	2	3		
	pcon	2	3		
	apex	2	3		
	ltex	2	4		
	tool	2	3		
	sced	1	3		
	cplx	5	6		
	KSLOC	75	125		
OSP2	prec	3	5	flex	3
	pmat	4	5	resl	4
	docu	3	4	team	3
	ltex	2	5	time	3
	sced	2	4	stor	3
	KSLOC	75	125	data	4
				pvol	3
				ruse	4
				rely	5
				acap	4
				pcap	3
				pcon	3
				apex	4
				plex	4
				tool	5
				cplx	4
				site	6

	float			fixed	
project	variable	low	high	variable	setting
flight	rely	3	5	tool	2
	data	2	3	sced	3
	cplx	3	6		
	time	3	4		
	stor	3	4		
	acap	3	5		
	apex	2	5		
	pcap	3	5		
	plex	1	4		
	ltex	1	4		
	pmat	2	3		
	KSLOC	7	418		
ground	rely	1	4	tool	2
	data	2	3	sced	3
	cplx	1	4		
	time	3	4		
	stor	3	4		
	acap	3	5		
	apex	2	5		
	pcap	3	5		
	plex	1	4		
	ltex	1	4		
	pmat	2	3		
	KSLOC	11	392		

Fig. 3. Four case studies.

Over the history of the COCOMO project, it has been observed that all the linear parameters that increase/decrease effort have the following slopes:

$$\begin{array}{c|c} increasing\ effort & decreasing\ effort \\ \hline 0.073 \leq m \leq 0.21 & -0.178 \leq m \leq -0.078 \end{array} \tag{2}$$

Similarly, the linear relations in the COQUALMO defect model linear relationships fall within very narrow slopes:

$$\begin{array}{l|c|c} phase & increasing\ defects & decreasing\ defects \\ \hline requirements & 0 \leq m \leq 0.112 & -0.183 \leq m \leq -0.035 \\ design & 0 \leq m \leq 0.14 & -0.208 \leq m \leq -0.048 \\ coding & 0 \leq m \leq 0.14 & -0.19 \leq m \leq -0.053 \end{array} \tag{3}$$

Like COCOMO, COQUALMO also includes scale factors that affect the estimates exponentially. These scale factors hinge about the origin and have the following slopes:

$$\begin{array}{l|c} phase & defect\ removal \\ \hline requirements & 0.08 \leq m \leq 0.14 \\ design & 0.1 \leq m \leq 0.156 \\ coding & 0.11 \leq m \leq 0.176 \end{array} \tag{4}$$

Clearly, it is possible to sample the space of all known COCOMO / COQUALMO tunings by picking random m values from Equation 2, Equation 3, and Equation 4.

To sample across the space of THREAT tunings, another mechanism is required. Tables like Figure 1 can be represented as an exponentially decaying function that peaks in one corner of the risk table at a value between two to four. Since this model is heuristic in nature, the exact height of the peak is not certain. When we perform tuning samplings over THREAT, we vary the height of the peak by a random factor $0.5 \leq x \leq 1$ if the peak is four, and $0.5 \leq x \leq 1.5$ if the peak is two.

4 Searching over Tuning Variables with Local Calibration

This paper compares estimates generated in two ways. This section describes local calibration (LC), the standard regression procedure used by the COCOMO community. Later, we will compare LC's results with those from SEESAW.

LC assumes that a matrix $D_{i,j}$ holds:

- The natural log of the LOC (lines of code) estimates;
- The natural log of the actual efforts for each project j;
- The natural logarithm of the cost drivers (the scale factors and effort multipliers) at locations $1 \leq i \leq 15$ (for COCOMO 81) or $1 \leq i \leq 22$ (for COCOMO-II).

With those assumptions, Boehm [8] shows that for COCOMO 81, the following calculation yields estimates for "a" and "b" that minimize the sum of the squares of residual errors:

$$\left.\begin{array}{rl} EAF_i &= \sum_j^N D_{i,j} \\ a_0 &= t \\ a_1 &= \sum_i^t KLOC_i \\ a_2 &= \sum_i^t (KLOC_i)^2 \\ d_0 &= \sum_i^t (actual_i - EAF_i) \\ d_1 &= \sum_i^t ((actual_i - EAF_i) * KLOC_i) \\ b &= (a_0 d_1 - a_1 d_0)/(a_0 a_2 - a_1^2) \\ a_3 &= (a_2 d_0 - a_1 d_1)/(a_0 a_2 - a_1^2) \\ a &= e^{a_3} \end{array}\right\} \quad (5)$$

In the case of COCOMO-I [8] these a, b values are used in the following equation to generate effort estimates. In this equation, EM_i are the effort multipliers from Figure 2:

$$effort = a \cdot KSLOC^b \cdot \prod_i^{15} EM_i$$

What is not widely appreciated is the size of the variance in the (a, b) values. The left-hand-side of Figure 4 shows the COCOMO-I (a, b) values learned by Baker [7] after, 300 times, extracting 10 projects at random from COCOMO data sets, then applying Equation 5 to the remaining data. The data sets used in this study contained 93 projects, so LC was applied to $\frac{93-10}{93} = 89\%$ of the data. A pre-experimental intuition was that we were using enough of the data to yield stable (a, b) values. As can be clearly seen by the wide variance on the (a, b) values in Figure 4, this was not the case.

The right-hand-side of Figure 4 shows the magnitude of the relative error (or MRE) values seen in Baker's study (MRE is a standard measure in the effort estimation field as

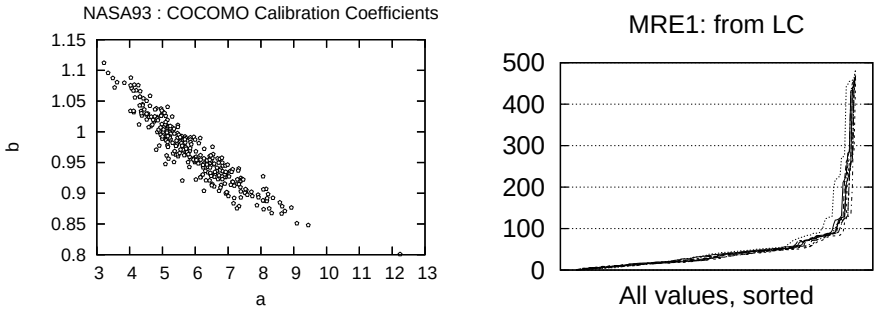

Fig. 4. Results of applying LC numerous times to 90% of the NASA93 data sets (available from `http://promisedata.org/data`). Left-hand-side shows computed (a, b) values. Right-hand-side shows MREs generated in 20 repeats over the NASA93 data set.

follows: $MRE = \frac{abs(actual - predicted)}{actual}$). In the sequel, we will refer to Baker's results as MRE1.

Some of the MRE1 errors are very large (up to nearly 500%) suggesting that LC was incomplete or that the variance in the (a, b) calculations have significant impact on the estimation. Note that this right-hand-side figure is not without precedent in the estimation literature: it is a well-established result that initial development effort estimates may be incorrect by a factor of four [8] or even more [24].

Elsewhere we have been partially successful in reducing estimation variance of Figure 4 using feature subset selection (FSS) [14, 32] or more data collection. Unfortunately, FSS reduces but does not eliminate the a, b variance. Also, further data collection is possible, but only at great organizational expense. Since we failed to generate precise tunings that yield exact estimates, we considered a change in our research goals. Perhaps, we argued, it was time to explore the space of possible tunings. This line of thinking led to the SEESAW search engine.

5 Searching over Project Choices with SEESAW

SEESAW seeks the smallest set of project choices that minimizes a score function that combines COCOMO's effort Ef and time Ti predictions; the COQUALMO defect De predictions; and the THREAT Th prediction. SEESAW normalizes these scores 0..100 then seeks ways to minimize the following combination of the normalized scores:

$$\sqrt{Ef^2 + Ti^2 + Th^2 + De^2} \tag{6}$$

(This is the Euclidean distance to minimum values for effort, time, threats, and defects.)

Figure 5 shows SEESAW's pseudo-code. Before exploring the low-level details of the algorithm (as done below), we first remark about the main purpose of the code. While SEESAW staggers over the space of possible tunings (see Equations 2, 3, and 4) and project choices (as defined by Figure 3), it only constrains project choices (and not

```
1  function run (AllRanges, ProjectConstraints) {
2    OutScore = -1
3    P = 0.95
4    Out = combine(AllRanges, ProjectConstraints)
5    Options = all Out choices with ranges low < high
6     while Options {
7       X = any member of Options, picked at random
8       {Low, High} = low, high ranges of X
9       LowScore = score(X, Low)
10      HighScore = score(X, High)
11       if LowScore < HighScore
12         then Maybe = Low; MaybeScore = LowScore
13         else Maybe = High; MaybeScore = HighScore
14       fi
15       if MaybeScore < OutScore  or P < rand()
16         then delete all ranges of X except Maybe from Out
17           delete X from Options
18           OutScore = MaybeScore
19       fi
20    }
21   return backSelect(Out)
22 }
23 function score(X, Value) {
24    Temp = copy(Out)  ;; don't mess up the Out global
25    from Temp, remove all ranges of X except Value
26    run monte carlo on Temp for 100 simulations, scoring each run using Equation 3.
27     return median score of the 100
28 }
```

Fig. 5. Pseudocode for SEESAW.

tuning variables). In this sense, SEESAW is the *opposite* of LC in that the latter makes no comment on the project choices. Rather, LC just proposes constraints to two tuning variables.

SEESAW is an adaption of Kautz & Selman's MaxWalkSat local search procedure [22]. Each solution is scored via a Monte Carlo procedure (see `score` in Figure 5), and SEESAW seeks to *minimize* that score (since, for our models, it is some combination of defects, development effort, development time, and threats).

SEESAW first combines the ranges for all the COCOMO project choices with the known project constraints of Figure 3. These constraints range from Low to $High$ values. If a case study does not mention a project choice, then there are no constraints on that choice, and the $combine$ function (line 4) returns the entire range of that choice. Otherwise, $combine$ returns only the values from Low to $High$.

In the case where a choice is $fixed$ to a single value, then $Low = High$. Since there is no decision to be made for this choice, SEESAW ignores it. The algorithm explores only those choices with a range of $Options$ where $Low < High$ (line 5). In each iteration of the algorithm, it is possible that one acceptable value for a choice X will be discovered. If so, the range for X is reduced to that single value, and the choice is not examined again (line 17).

SEESAW prunes the final recommendations (line 21). This function removes the N selections added last that do not significantly change the final score (t-tests, 95% confidence). This culls any final irrelevancies in the selections.

The $score$ function shown at the bottom of Figure 5 calls COCOMO / COQUALMO / THREAT models 100 times, each time selecting:

- Random values for the project choices (from the $Options$ set);
- Random values for the tuning variables (as described in Equations 2, 3, and 4).

The median value of the Equation 6 values seen in these runs is the $score$ for those project choices. As SEESAW executes, the ranges in $Options$ are removed and replaced by single values (lines 16-17), thus constraining the space of possible simulations.

SEESAW was designed after observing experimentally that the most interesting ranges in $Options$ are generally the minimum and maximum values. The reason for this is simple: All the functions in COCOMO / COQUALMO / THREAT are monotonic, causing the most dramatic effects to occur at the extreme ends of the ranges. In fact, SEESAW takes its name from the way earlier versions of this algorithm tended to seesaw between extreme values. We have conducted experiments with other approaches that allow intermediate values. On comparison with the simulated annealing method used in a prior publications [31], we found that seesawing between $\{Low, High\}$ values was adequate for our purposes.

SEESAW is a stochastic algorithm: the selection of the next choice to explore is completely random (line 7). We use this stochastic approach since much research from the 1990s showed the benefit of such search methods. Not only can stochastic algorithms solve non-linear problems and escape from local minima/maxima, but they can also find solutions faster than complete search, and for larger problems [34]. For example, we have implemented a deterministic version of SEESAW that replaces the random selection of *one* choice in line 7 with a search through *all* choice for the best $\{Low, High\}$ value. That algorithm ran much slower (runtimes were 12 times greater) with nearly identical results to those of the stochastic search. Crawford and Baker [16] offer one explanation for the strange success of stochastic search. For models where the solutions are a small part of the total space, a complete search wastes much time exploring uninformative areas of the problem. A stochastic search, on the other hand, does not get stuck in such uninformative areas.

6 Experiments

In terms of trusting SEESAW, the key question is how much SEESAW"S estimates differ from those generated by conventional methods such as LC. To implement that comparison, we followed the procedure shown on the right-hand-side of Figure 6.

1. Two kinds of control policies were explored; i.e., the strategic and tactical choices marked in Figure 2.
2. Five kinds of projects were created; i.e., the four projects from Figure 3 and one using an imaginary project whose project choices included the entire range of all COCOMO variables.
3. For these 2*5=10 case studies, we ran SEESAW to find the constraints that led to minimum effort, threats, defects, and development time.
4. From each of the 10 sets of constraints, we generated projects consistent with those constraints.

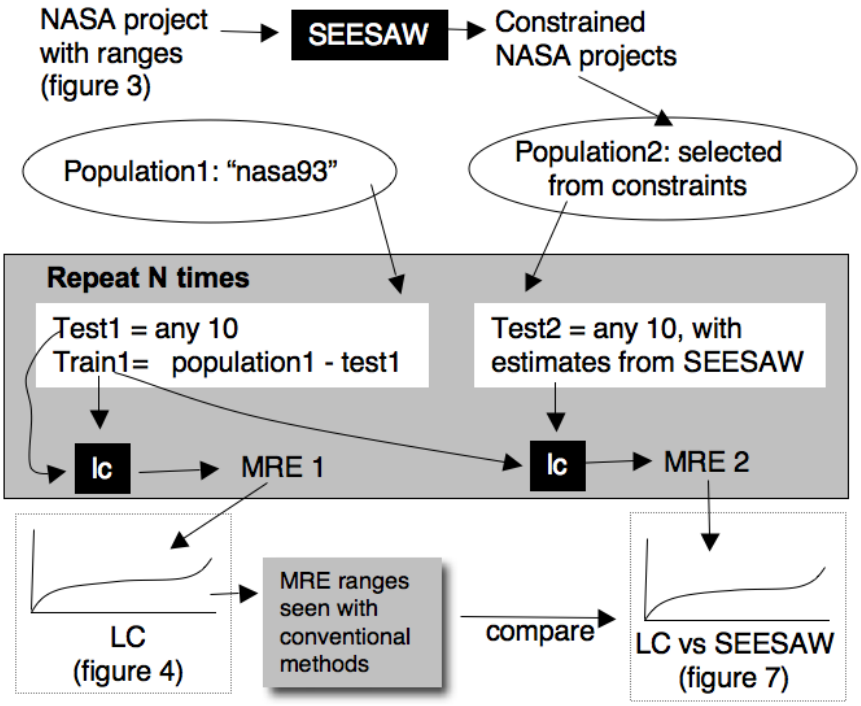

Fig. 6. Experimental design.

5. Estimates were then added to the above randomly generated projects. Each estimate was the median value seen in 1000 simulations of SEESAW's `score` function (see line 23 of Figure 5). That is, these estimates were generated from constrained project choices but unconstrained tuning variables.
6. SEESAW's predicted effort estimates were then compared to those generated by conventional means; i.e., LC learning on NASA93, then applied to the projects generated from the constraints found by SEESAW.
7. The delta in SEESAW's and LC's estimates was computed using $\Delta = \frac{abs(SEESAW - LC)}{SEESAW}$.
8. Steps 4, 5, 6, and 7 were repeated 20 times to generate the set "MRE2".

Figure 7 shows the sorted MRE2 values for 10 sets of projects (one line for each project). The median deltas Δ seen in these ten cases studies runs were

$$\Delta = \frac{abs(SEESAW - LC)}{SEESAW} = \{20, 20, 21, 22, 23, 23, 23, 23, 24, 24, 26\}\%$$

We explain the small median delta as follows: the project choices found by SEESAW forced this process model into a narrow space of behaviors. In this narrow space, the impact of the tuning variance becomes unimportant.

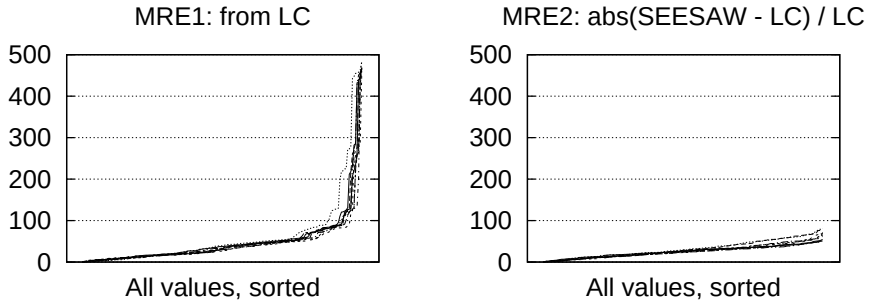

Fig. 7. MRE2 results for Figure 6, for ten case studies (one study per line). The MRE1 results (left-hand-side) come from Figure 1.

But there is a problem with this claim that a delta Δ value of 20 to 26 is a "small delta." Any claim that that X% is "small" is spurious unless we have some other study that says "in conventional estimation, it is uncommon to see variance smaller than this." Accordingly, we return to the Baker's LC results. Figure 7 displays the Baker results (MRE1: left-hand-side) and the delta Δ values generated above (MRE2: right-hand-side). A visual inspection shows that the MREs generated via local calibration (MRE1) are mostly the same size as the the differences seen between LC's and SEESAW's predictions (MRE2). Indeed, in a small number of cases, the MRE1 values are much larger values (see the spike, right-hand-side of the MRE1 distribution).

For the sake of completeness, we performed a statistical difference test on the MRE1 and MRE2 distributions:

- Since we are comparing distributions, paired tests were deemed inappropriate.
- Since Figure 4 and Figure 7 have a small number of large outliers, tests that make a Gaussian assumption were also deemed inappropriate.
- Therefore, we compared Figure 4 (right-hand-side) with the ten distributions of Figure 7 using Mann-Whitney tests. At the 99% confidence level, the distributions were indistinguishable in five out of ten of our experiments.

This test confirms our visual impression: the MRE1 distributions (obtained by controlling the tuning variables) are not always greatly different to MRE2 distributions (obtained without controlling the tuning variables).

These results do *not* say that SEESAW produces the same estimates as LC. Clearly, SEESAW's Monte Carlo approach to estimation does introduce some "jitter" into the estimates. The question is, is that jitter significantly large? We would argue that it is not significantly large. As mentioned above, the constraints on the project choices learned by SEESAW have forced these models into a narrow space of behaviors. In that narrow space, the impact of the tuning variance becomes unimportant:

- Figure 7 shows visually that that the jitter associated with SEESAW (MRE2) is close to, and sometimes smaller than, the jitter produced by merely changing the training sample by 10% (MRE1).

– Our Mann-Whitney tests confirm this visual impression.

We conclude that the estimates generated after constraining project choices (using SEESAW) are no more varied than the estimates generated after constraining tuning variables (using LC).

7 Conclusion

In prior studies with the COCOMO / COQUALMO / THREAT models, we have shown that an AI search engine can find a minimal set of constraints to project choices that reduce estimates for project effort, defects, time, and threats [30, 31]. Those results were based on limited case studies and failed to check the validity of the estimates.

In this report, we showed in ten case studies that the deltas between ranges of estimates generated by SEESAW (without constraining the tuning variables) form a similar distribution to those found by traditional estimation methods (that constrain the tuning variables). In fact, in half our experiments, there was no statistically significant delta between the two distributions. From a technical perspective, this means that if estimation variance arises from a tuning variance T and project variance P, then there exists process models such as COCOMO / COQUALMO where $P \gg T$; i.e., project choices dominate tuning options.

When $P \gg T$, the estimates found by constraining project choices will be close to estimates found via tuning on historical data. For such models, tuning is optional and decision making need not wait for detailed local data collection.

8 Future Work

This work could lead to a "Goldilocks" principle for process modeling:

- Very small models offer trite conclusions that are insensitive to important project features.
- Very large models need extensive data collection to tune their internal structures.
- In between there may exist some models that are "just right"; i.e., big enough to make interesting conclusions, but small enough such that the internal tuning variance does not dominate the variance results from input project choices.

We have presented here evidence that the COCOMO / COQUALMO / THREAT models are "just right"; i.e., their variance can be reduced by constraining the project choices while leaving the tuning variables unconstrained.

We make no claim that *all* process models are "just right" and, hence, can be controlled by SEESAW. Indeed, we suspect that many process models may not be near the right size. However (and this is the main point of this paper), what kind of process model should be used? We would argue, that for data-starved domains, we should *deliberately select* for "just right" process models.

Based on the above, the future direction of this research is clear: for data-starved domains, derive better design principles for process models. Our intuition is that this

will be a search problem and that tools like SEESAW will not only be useful for searching a fixed model, but also for finding revisions to current models in order to make them "just right".

References

1. T. Abdel-Hamid and S. Madnick. *Software Project Dynamics: An Integrated Approach*. Prentice-Hall Software Series, 1991.
2. Jesus S. Aguilar-Ruiz, Isabel Ramos, Jose Riquelme, and Miguel Toro. An evolutionary approach to estimating software development projects. *Information and Software Technology*, 43(14):875–882, 2001.
3. M. Akhavi and W. Wilson. Dynamic simulation of software process models. In *Proceedings of the 5th Software Engineering Process Group National Meeting (Held at Costa Mesa, California, April 26 - 29)*. Software engineering Institute, Carnegie Mellon University, 1993.
4. J. L. Alvarez, J. Mata, Jose C. Riquelme, and I. Ramos. A data mining method to support decision making in software development projects. In *ICEIS'2003: Fifth International Conference on Enterprise Information Systems*, 2003.
5. J.H. Andrews, F.C.H. Li, and T. Menzies. Nighthawk: A two-level genetic-random unit test data generator. In *IEEE ASE'07*, 2007. Available from `http://menzies.us/pdf/07ase-nighthawk.pdf`.
6. John Bailey. Using monte carlo and cocomo-2 to model a large it system development, 2002.
7. Dan Baker. A hybrid approach to expert and model-based effort estimation. Master's thesis, Lane Department of Computer Science and Electrical Engineering, West Virginia University, 2007. Available from `https://eidr.wvu.edu/etd/documentdata.eTD?documentid=5443`.
8. B. Boehm. *Software Engineering Economics*. Prentice Hall, 1981.
9. B. Boehm. Safe and simple software cost analysis. *IEEE Software*, pages 14–17, September/October 2000. Available from `http://www.computer.org/certification/beta/Boehm_Safe.pdf`.
10. Barry Boehm, Ellis Horowitz, Ray Madachy, Donald Reifer, Bradford K. Clark, Bert Steece, A. Winsor Brown, Sunita Chulani, and Chris Abts. *Software Cost Estimation with Cocomo II*. Prentice Hall, 2000.
11. L.C. Briand, T. Langley, and I. Wieczorek. A replicated assessment and comparison of common software cost modeling techniques. In *Proceedings of the 22nd International Conference on Software Engineering, Limerick, Ireland*, pages 377–386, 2000.
12. Lionel C. Briand, Khaled El Emam, and Frank Bomarius. Cobra: A hybrid method for software cost estimation, benchmarking, and risk assessment. In *ICSE*, pages 390–399, 1998.
13. A.G. Cass, B. Staudt Lerner, E.K. McCall, L.J. Osterweil, Stanley M. Sutton Jr., and A. Wise. Little-jil/juliette: A process definition language and interpreter. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000)*, pages 754–757, June 2000.
14. Zhihoa Chen, Tim Menzies, and Dan Port. Feature subset selection can improve software cost estimation. In *Proceedings, PROMISE workshop, ICSE 2005*, 2005. Available from `http://menzies.us/pdf/05/fsscocomo.pdf`.
15. S. Chulani, B. Boehm, and B. Steece. Bayesian analysis of empirical software engineering cost models. *IEEE Transaction on Software Engineerining*, 25(4), July/August 1999.
16. J. Crawford and A. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *AAAI '94*, 1994.
17. N. E. Fenton and M. Neil. A critique of software defect prediction models. *IEEE Transactions on Software Engineering*, 25(5):675–689, 1999. Available from `http://citeseer.nj.nec.com/fenton99critique.html`.

18. D. Ferens and D. Christensen. Calibrating software cost models to Department of Defense Database: A review of ten studies. *Journal of Parametrics*, 18(1):55–74, November 1998.
19. D. Harel. Statemate: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16(4):403–414, April 1990.
20. O. Jalali, T. Menzies, Omid Jalali, M. Feather, and J.Kiper. Real-timee requirements engineering, 2008. Available from `http://menzies.us/pdf/08realtime.pdf`.
21. R. Jensen. An improved macrolevel software development resource estimation model. In *5th ISPA Conference*, pages 88–92, April 1983.
22. H. Kautz, B. Selman, and Y. Jiang. A general stochastic approach to solving problems with hard and soft constraints. In D. Gu, J. Du, and P. Pardalos, editors, *The Satisfiability Problem: Theory and Applications, New York, NY*, pages 573–586, 1997. Available on-line at `http://citeseer.ist.psu.edu/168907.html`.
23. D. Kelton, R. Sadowski, and D. Sadowski. *Simulation with Arena, second edition*. McGraw-Hill, 2002.
24. C.F. Kemerer. An empirical validation of software cost estimation models. *Communications of the ACM*, 30(5):416–429, May 1987.
25. A. Law and B. Kelton. *Simulation Modeling and Analysis*. McGraw Hill, 2000.
26. K. Lum, J. Powell, and J. Hihn. Validation of spacecraft software cost estimation models for flight and ground systems. In *ISPA Conference Proceedings, Software Modeling Track*, May 2002.
27. Karen Lum. Software cost analysis tool user document, 2005.
28. Karen Lum, Michael Bramble, Jairus Hihn, John Hackney, Mori Khorrami, and Erik Monson. Handbook for software cost estimation, 2003.
29. R.H. Martin and D. M. Raffo. A model of the software development process using both continuous and discrete models. *International Journal of Software Process Improvement and Practice*, June/July 2000.
30. T. Menzies, O. Elrawas, D. Baker, J. Hihn, and K. Lum. On the value of stochastic abduction (if you fix everything, you lose fixes for everything else). In *International Workshop on Living with Uncertainty (an ASE'07 co-located event)*, 2007. Available from `http://menzies.us/pdf/07fix.pdf`.
31. T. Menzies, O. Elwaras, J. Hihn, Feathear nd B. Boehm M, and R. Madachy. The business case for automated software engineerng. In *IEEE ASE*, 2007. Available from `http://menzies.us/pdf/07casease-v0.pdf`.
32. Tim Menzies, Zhihao Chen, Jairus Hihn, and Karen Lum. Selecting best practices for effort estimation. *IEEE Transactions on Software Engineering*, November 2006. Available from `http://menzies.us/pdf/06coseekmo.pdf`.
33. P. Mi and W. Scacchi. A knowledge-based environment for modeling and simulation software engineering processes. *IEEE Transactions on Knowledge and Data Engineering*, pages 283–294, September 1990.
34. R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995. (reprinted 1997,2000).
35. R. Park. The central equations of the price software cost model. In *4th COCOMO Users Group Meeting*, November 1988.
36. Parag C. Pendharkar, Girish H. Subramanian, and James A. Rodger. A probabilistic model for predicting software development effort. *IEEE Trans. Softw. Eng.*, 31(7):615–624, 2005.
37. L. Putnam and W. Myers. *Measures for Excellence*. Yourdon Press Computing Series, 1992.
38. D. Raffo and T. Menzies. Evaluating the impact of a new technology using simulation: The case for mining software repositories. In *Proceedings of the 6th International Workshop on Software Process Simulation Modeling (ProSim'05)*, 2005.
39. D. M. Raffo, J. V. Vandeville, and R. Martin. Software process simulation to achieve higher cmm levels. *Journal of Systems and Software*, 46(2/3), April 1999.

40. D.M. Raffo. Modeling software processes quantitatively and assessing the impact of potential process changes of process performance, May 1996. Ph.D. thesis, Manufacturing and Operations Systems.
41. L. Rela. Evolutionary computing in search-based software engineering. Master's thesis, Lappeenranta University of Technology, 2004.
42. A. Wise, A.G. Cass, B. Staudt Lerner, E.K. McCall, L.J. Osterweil, and Jr. S.M. Sutton. Using little-jil to coordinate agents in software engineering. In *Proceedings of the Automated Software Engineering Conference (ASE 2000) Grenoble, France.*, September 2000. Available from `ftp://ftp.cs.umass.edu/pub/techrept/techreport/2000/UM-CS-2000-045.ps`.