Accurate Estimates Without Local Data?

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. Models of software projects input project details and output predictions via their internal tunings. The output predictions, therefore, are affected by variance in the project details P and variance in the internal tunings T. Local data is often used to constrain the internal tunings (reducing T).

While constraining internal tunings with local data is always the preferred option, there exist some models for which constraining tuning is optional. We show empirically that, for the USC COCOMO family of models, the effects of P dominate the effects of T i.e. the output variance of these models can be controlled *without using local data to constrain the tuning variance* (in ten case studies, we show that the estimates generated by only constraining P are very similar to those produced by constraining T with historical data).

We conclude that, if possible, models should be designed such that the effects of the project options dominate the effects of the tuning options. Such models can be used for the purposes of decision making without elaborate, tedious, and time-consuming data collection from the local domain.

KEYWORDS: AI, decision making, software engineering, model-based project management, search.

LENGTH: 5500 words + 8 figures (at 250 words each) = 7500 words

1 Introduction

Predictions are generated using data and/or models. If data is available, it is possible to learn new models. If data is unavailable, it is possible to reuse models learned from

other sites. Typically, models taken from other sites must be tuned using local data. This paper is about a novel technique for reusing models *without* tuning them.

In model-based project management, models are used to find better ways to organize a project. Managers conduct what-if queries across the space of project options P to find a subset of those options that most improves predictions about the project (e.g. reduce defects and development time). This problem can be formalized as follows: find the smallest $P' \subseteq P$ that most improves model estimates. Conceptually, the estimate generated by a model are a function of the project options P and the internal model tunings T; i.e.

$$\textit{estimate} = \textit{project} * \textit{tuning} \tag{1}$$

For example, consider the following simplified COCOMO model,

$$effort = a \cdot LOC^{b+pmat} \cdot acap \tag{2}$$

Here, a, b control the linear and exponential effects (respectively) on model estimates. while $pmat$ (process maturity) and $acap$ (analyst capability) are project choices adjusted by managers. That is, the tuning options T are the range of a, b and the project options P are the range of $pmat$ and $acap$.

Traditional approaches use historical data to reduce the space of possible calibrations (e.g. using regression). In our approach, we leave the tunable variables unconstrained and instead use an AI search engine to reduce the space of possibilities in the project options. Our SEESAW tool performs large scale what-if queries looking for the smallest $P' \subseteq P$ that most improves model estimates. SEESAW constrains project options P, *but not the tuning options* T. In two studies (ASE 2007 [27] and ICSE 2009 [28]) Menzies, Boehm, Hihn, et al. found that, at least for the USC models we use, estimate variance was more controlled by project variance. That is, in terms of what factors dominate the output:

$$\textit{estimate} = \textit{project} * \textit{tuning} \tag{3}$$

Note the *reverse case* of the above equation:

$$\textit{estimate} = \textit{project} * \textit{tuning} \tag{4}$$

In the case of Equation 4, the tuning variance is the major controller of the estimate and decision making must be delayed until after tuning. Equation 4 is undesirable when it is difficult to access the data required for constraining model tunings. This "data drought" is quite common:

– Metrics-guru Norman Fenton spent years advocating careful data collection [18]. Recently, he has despaired of that approach. At a keynote address in 2007[5] he shocked his audience by saying:

> *"....much of the current software metrics research is inherently irrelevant to the industrial mix ... any software metrics program that depends on some extensive metrics collection is doomed to failure."*

– Our experience is similar to that of Fenton. 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 [27].

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. Also, within the organizations we have contact with, we note a decreasing to share data compared to, say, 15 years ago.

Whatever the reason, we often need to adjust our modeling methodologies to accommodate the data drought. For example, we could assume Equation 3 and use SEESAW. But before we can trust those tools we must test that AI tools like SEESAW not less accurate at estimation than (say) traditional linear regression methods. Accordingly, this paper compares:

– Estimates generated by conventional methods that reduce the tuning variance T using local data;
– Estimates generated by SEESAW that reduce the project variance P without constraining the tuning variance.

For the USC models used in this study, the range of estimate errors seen after constraining the project options P (but not the tuning options T) is similar to the range seen after constraining just the tuning options T. That is, using our AI methods, we can perform model-based project management without requiring local data.

The rest of this paper is structured as follows. After discussing the implications of this 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 model estimates after constraining (a) just the project options using SEESAW or (b) just the tuning options using linear regression.

[5] `http://promisedata.org/?cat=130`

2 Implications

2.1 Implications of SEESAW for Early Lifecycle Decision Making

Standard practice in model building normally involves three steps:

- *Step 1:* Collecting domain knowledge (previous results, expert knowledge);
- *Step 2:* Building a initial model based on step 1 including as yet unknown parameters. Note that these unknowns represent a range of tuning options.
- *Step 3*: Constraining tuning options using local data (e.g. via linear regression).

In domains suffering from a data drought, we cannot consider methods requiring large volumes of data. In the approach advocated by this paper, we reuse existing models (from USC) and then study them with an AI search engine that only reduces project options (and not tuning options). This approach removes the need for Step 1 and Step 2, and does not require Step 3. Yet, as shown below, it still makes accurate effort predictions. Since our method does not require detailed domain knowledge it can be applied very early in the process life cycle.

From a business perspective, our result means that certain 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 variance 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 when it is difficult to find relevant data within a single organization to constrain the tuning options of a model.

2.2 Implications of the "Data Drought" on Model Selection

Our introduction mentioned Norman Fenton's pessimism on the practicality of industrial data collection for software engineering. Sometimes, Fenton's pessimism is unfounded. There exists an increasing number of high process maturity organizations that store large amounts of consistent data collected from projects with well defined processes. The techniques reported in this paper (AI search using the SEESAW algorithm) are not required for such data-rich domains.

However, in data-starved domains, a "Goldilocks" principle might be appropriate:

- Tiny models offer trite conclusions and are insensitive to important project features.

– Very large models need extensive data collection to constrain the internal tunings.
– In between there may exist some models that are "just right"; i.e., big enough to draw interesting conclusions, but small enough such that the internal tuning variance does not dominate the variance results from input project options.

We make no claim that *all* process models are "just right" and, hence, can be controlled by our methods. Such process models can be quite complex and include: discrete-event models [23, 25]; system dynamics models [1]; state-based models [3, 19, 26]; rule-based programs [30]; or standard programming constructs such as those used in Little-JIL [11, 36]. These rich modeling frameworks allow the representation of detailed insights into an organization. However, in data starved domains, the effort required to tune may be non-trivial. In terms of the Goldilocks principle, we suspect that many process models may not be near the "right size" and will require extensive tuning before they can be used for decision making. Hence, in domains suffering from a data drought, we would advocate "just right" models like those from USC.

3 Tools Used in Our Experiments

3.1 Models used in this Study

The USC models, we argue, are "just right" because they have been developed, refined, and constrained over a very long period of time. The range of tuning options we explore below are taken from 30 years of modeling experience and regression studies of hundreds of projects [9]. The variables in our models have been selected and tested by a large community of academic and industrial researchers led by Boehm (this large group has meet annually since 1985). It is hardly surprising that, for the USC models, the project options dominate the tuning options since these tuning options have been refined and constrained by decades of work.

The USC models have other useful features. Unlike other models such as PRICE TRUE PLANNING [32], SLIM [34], 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 [13]. Further, we work extensively with government agencies writing software. Amongst those agencies, these models are frequently used to generate and justify budgets. Lastly, the space of possible tunings within COCOMO & COQUALMO is well defined (see below). Hence, it is possible to explore the space of possible tunings.

[Fig. 1 about here.]

Figure 1 shows the variables used by our models. The last two columns of this figure show the results of a Delphi panel session at the Jet Propulsion Laboratory (JPL) 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. When searching for subsets of project options that improve model predictions, we take care to separate our search into either *tactical* or *strategic* runs. Note that these definitions of $strategic$ and $tactical$ options are not hard-wired into our system. If a user disagrees with our definitions of strategic/tactical, they can change a simple configuration file.

This study uses three USC models:

1. The COQUALMO software *defect* predictor [10, p254-268]. COQUALMO models defect introduction and defect removal in requirements, design, and coding.
2. The COCOMO *effort* and development *time* predictor [10, p29-57]. COCOMO assumes that effort depends exponentially on some *scale factors* and linearly to some *effort multipliers*. COCOMO estimates development work months (total) and calendar months (elapsed) and includes all coding, debugging, and management activities.
3. The THREAT model [10, p284-291] contains a large set of two-dimensional tables like Figure 4 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).

When searching for good subsets of the project options, our AI search engine tries to minimize the outputs of these above models, combined as follows:

$$combined = \sqrt{Effort^2 + Time^2 + Threats^2 + Defects^2} \tag{5}$$

This is the Euclidean distance to minimum values for all these predictions. Note one small technical detail: in order that one measure does not dominate over the others, we normalize all predictions to the range 0..100.

3.2 P: The Project Options

Figure 2 summarizes four NASA case studies using the project options of Figure 1:

- "OSP" is the GNC (guidance, navigation, and control) component of NASA's 1990s *Orbital Space Plane*;
- "OSP2" is a later version of OSP;
- "Flight" and "ground" show typical ranges of NASA's Jet Propulsion Laboratory.

[Fig. 2 about here.]

Inside our model, project options typically range from 1 to 5 where "3" is the nominal value that offers no change to the default estimate. Some of the project options in Figure 2 are known precisely (see all the options with single *values*). But many of the features in Figure 2 do not have precise values (see all the features that *range* from some *low* to *high* value).

Sometimes the ranges of options 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 2 does not mention all the features listed in Figure 1 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 options not mentioned in Figure 2, values are picked at random from the full range of Figure 1.

3.3 T : the Tuning Options

Many of our project options 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:

$$estimate = m(x - 3) + 1 \tag{6}$$

Over the history of the COCOMO project, it has been observed that all the linear parameters that increase/decrease effort have the slopes of Figure 3a. Similarly, the linear relations in the COQUALMO defect model linear relationships fall within very narrow slopes of Figure 3b.

Like COCOMO, COQUALMO also includes scale factors that affect the estimates exponentially. These scale factors hinge about the origin and have the slopes of Figure 3c.

[Fig. 3 about here.]

Note also that COCOMO includes two other tuning points: the $\{a, b\}$ "calibration parameters" that can any linear and exponential effects: see Equation 8, discussed below.

Sampling across Equation 6 is a simple matter of picking random m values from Figure 3. Similarly, it is possible to sample the space of $\{a, b\}$ values by selecting at random from their known ranges (see Figure 5, below).

To sample across the space of THREAT tunings, another mechanism is required. Tables like Figure 4 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.

[Fig. 4 about here.]

3.4 Changing Project Options P with SEESAW

SEESAW seeks the smallest subset of the project options $P' \subseteq P$ that most improves model estimates of Equation 5. To this, it sets one variable at a time (selected at random) using a greedy stochastic search (see the algorithm described in the appendix). The resulting estimates (combined using Equation 5) are then logged. At the end, SEESAW reviews the effects of how setting the first, second, third (etc) variables changes the resulting predictions. SEESAW recommends all the settings up to the setting where the resulting estimates are all minimized. That is, SEESAW be can viewed as *either* a tool for selecting the best $P' \subseteq P$ *or* as a tool for pruning away irrelevant project options (those not found in P').

Note that:

- One run of SEESAW selects at random from the tuning options (see Equations Figure 3) and project options (as defined by Figure 2).
- SEESAW constrains project options P and not the tuning options T.

3.5 Changing Tuning Options T with "LC"

At the end of this paper is an experiment comparing the effects of generating estimates after reducing project options P with SEESAW or reducing tuning options T. This

section describes "LC", the standard regression procedure used by the COCOMO community since 1981. LC changes the tuning parameters T inside the COCOMO model using local data.

LC assumes that some 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-I) or $1 \leq i \leq 22$ (for COCOMO-II).

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

$$\left.\begin{array}{rl} EAF_i &= \sum_j^N D_{i,j} \\ a_0 &= t \\ a_1 &= \sum_i^t KLOC_i \\ a_2 &= \sum_i^t (KLOC_i)^2 \\ d_0 &= \sum_i^i (actual_i - EAF_i) \\ d_1 &= \sum_i^i ((actual_i - EAF_i) * KLOC_i) \\ b &= (a_0 d_1 - a_1 d_0)/(a_0 a_2 - a_1^2) \\ a_3 &= (a_2 d_0 - a_1 d_1)/(a_0 a_2 - a_1^2) \\ a &= e^{a_3} \end{array}\right\} \quad (7)$$

Note that LC is the *opposite* of SEESAW in that the former makes no comment on the project options. Rather, LC just proposes constraints to two tuning variables (a, b).

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 1:

$$effort = a \cdot KSLOC^b \cdot \prod_i^{15} EM_i \quad (8)$$

Even after applying LC, the tuning variance can still be quite large. The left-hand-side of Figure 5 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 7 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 not using enough data to yield stable (a, b) values. As can be clearly seen by the wide variance on the (a, b) values in Figure 5, this was not the case.

[Fig. 5 about here.]

The right-hand-side of Figure 5 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

follows: $MRE\% = 100 * \frac{abs(actual-predicted)}{actual}$). 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 has 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 5 using feature subset selection (FSS) [12, 29] or more data collection. Unfortunately, further data collection is possible, but only at great organizational expense. Also, FSS reduces but does not eliminate the a, b variance. Since we failed to reduce estimation variance by constraining the tuning variables, we took another approach: we developed the SEESAW system to explore the effects of just constraining the projects variables.

4 Experiments

4.1 Goals

In terms of trusting SEESAW, the key question is how much SEESAW'S estimates differ from those generated by conventional methods such as LC. Formally, this can be expressed as two hypotheses:

H0: The effort estimates generated after constraining the tuning options T are different to those generated after *not* constraining them.

H1: The effort estimates generated by constraining just the project options P are similar to those seen after constraining just the tuning options T.

Having stated our goal formally in this way, we add that the following conclusions will be not be based solely on statistical significance tests of H0 vs H1. Cohen [14] is scathing about such tests when he writes that the significance hypothesis testing of H0 vs H1 is a "potent but sterile intellectual rake who leaves ... no viable scientific offspring". In support of Cohen's thesis, we offer the following salutary lesson. Writing in the field of marketing, Armstrong [6] reviews one study that, using significance testing, concludes that estimates generated from multiple sources do no better than those generated from a single source. He then demolishes this conclusion by listing 31 studies where multiple source prediction consistently out-performs single source prediction

by 3.4% to 23.4% (average=12.5%). These improvements, in every study surveyed by Armstrong, are the exact opposite of what would be predicted from the significance test results.

Accordingly, in the following, we will use simple visualizations to assess H0 vs H1 and demote significance tests to the role of a "reasonableness test" for the conclusions drawn from the visualizations.

4.2 Method

[Fig. 6 about here.]

This experimental design described below (and illustrated in Figure 6) lets us reflect over two distributions:

- MRE1: the difference between expected and actual when the tuning options T were changed by LC;
- MRE2: the difference between the predictions generated by LC or SEESAW.

Note that LC changes only the tuning options T while SEESAW changes project options P. Hence, by comparing the two distributions, we can assess H0 vs H1.

Our experiment executes as follows:

1. 5*2=10 different case studies were created:
   - Two kinds of control policies were explored; i.e., the strategic and tactical choices marked in Figure 1.
   - Five kinds of projects were created; i.e., the four projects from Figure 2 and one using an imaginary project whose project choices included the entire range of all COCOMO variables.
2. For these 2*5=10 case studies, we ran SEESAW to find the constraints that led to minimum effort, threats, defects, and development time.
3. From each of the 10 sets of constraints, we generated projects consistent with those constraints.
4. Effort estimates were then added to the above randomly generated projects. Each estimate was the median estimate value seen in 1000 simulations of SEESAW, for a particular set of project constraints. That is, these estimates were generated from constrained project choices but unconstrained tuning variables.

5. SEESAW's predicted effort estimates were then compared to those generated by conventional means; i.e., LC learning on some historical NASA data[6] then applied to the projects generated from the constraints found by SEESAW.
6. The delta in SEESAW's and LC's estimates was computed using $\Delta = 100 * \frac{abs(SEESAW - LC)}{SEESAW}$.
7. Steps 4, 5, 6, and 7 were repeated 20 times to generate the set "MRE2".

4.3 Results

[Fig. 7 about here.]

Figure 7, shows the MRE values seen in 10 case studies (one per line). The left-hand-side shows the sorted MRE1 results from LC and has median values

$$median(MRE) = \{25, 26, 27, 28, 29, 30, 31, 35, 37, 38\}\%$$

The right-hand-side shows the sorted MRE2 results and has median values

$$median(\Delta) = \{20, 20, 21, 22, 23, 23, 23, 23, 24, 25\}\%$$

Three features of Figure 7 are noteworthy:

- The SEESAW errors (in MRE2) are never greater than the LC errors (in MRE1).
- SEESAW's MRE2 errors are small (within 25% or less of the LC results). This is a surprising result, given that SEESAW samples effort from the unconstrained space of possible tunings. The only explanation for this effect is that, within the USC models, the project options P dominate the tuning options T. 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.
- The median differences between SEESAW and LC, as shown in MRE2, are quite small when compared to the range of values seen with just LC (MRE1). These deltas are very small when superimposed over the MRE1 curve (which can range to over 400%).

That is, a visual inspection of these results favors H1.

[6] NASA93, available for download at `http://promisedata.org/repository/data/nasa93/nasa93.arff`.

4.4 Significance Tests

For the sake of completeness, we performed a statistical significance test on the MRE1 and MRE2 distributions. Recalling Cohen and Armstrong (discussed above), we believe that such significance tests should be a secondary, not primary, evaluation criteria. For example, we use them as a reasonableness check on conclusions gathered via intuitive visual means.

- Since we are comparing distributions, paired tests were deemed inappropriate.
- Since Figure 5 and Figure 7 have a small number of large outliers, tests that make a Gaussian assumption were also deemed inappropriate.
- Therefore, we compared the left and right sides of Figure 7.
- In this case, Mann-Whitney tests did not violate our visual impressions. At the 99% confidence level, the distributions were different only in the minority case (four out of ten).

4.5 Summary of Results

In summary, the visualizations of Figure 7 make us endorse H1 (and statistical significance tests do not challenge that endorsement). Hence, we say that the estimates generated after constraining project options P (using SEESAW) are about the same as those generated after constraining tuning options T (using LC).

5 Related Work

To the best of our knowledge, this work is the first to try controlling the $project$ options while leaving the $tuning$ options unconstrained. Much of the related work on uncertainty in software engineering uses a Bayesian analysis. For example, Pendharkar et.al. [33] demonstrate the utility of Bayes networks in effort estimation while Fenton and Neil explore Bayes nets and defect prediction [16, 17] (but 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.

Even in the field in *search-based software engineering* (SBSE), we have not seen anything like this study. 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]. A recent review of 123 search-based SE papers [35] showed that much of that work relates to testing (e.g., simulated annealing 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.

The SBSE literature initially inspired us to try simulated annealing to search the what-ifs in untuned COCOMO models [27]. However, we found that SEESAW ran much faster and produced results with far less variance than simulated annealing.

As to our own work, for years, we have struggled with the data drought problem and have recommended elaborate feature subset selection methods to prune uninformative data [12, 29]. Here, we take a radically different approach and explore the value of models whose internal tunings have not been constrained with local data.

6 Conclusion

We are concerned with using models in domains suffering from a data drought. If a particular domain is data rich, the techniques described in this paper are not required. Otherwise, AI search methods like SEESAW can be used to reuse models from other sites *without* requiring local data.

To make that case, this report showed ten case studies comparing the deltas between ranges of estimates generated by SEESAW (without constraining the tuning options) and those found by traditional estimation methods (that constrain the tuning options). The distributions of the two estimates was found to be similar. From a technical perspective, this means that if estimation variance arises from a tuning variance T and project variance P, then there exist 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:

- Using local data to reduce tuning variance is optional;
- It is possible to compute accurate estimates without local data;
- Decision making need not wait for detailed local data collection.

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. J. S. Armstrong. Significance tests harm progress in forecasting. *Internatioal Journal of Forecasting*, 23:21–327, April 2007.
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. 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.
12. 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`.
13. 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.
14. J. Cohen. The earth is round (p ¡ .05). *American Psychologist*, 49:997–1003, 1988.

15. J. Crawford and A. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *AAAI '94*, 1994.
16. N. Fenton, M. Neil, W. Marsh, P. Hearty, Lukasz Radlinski, and P. Krause. *Empirical Software Engineering*, Oct 2008.
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. N. E. Fenton and S.L. Pfleeger. *Software Metrics: A Rigorous & Practical Approach (second edition)*. International Thompson Press, 1995.
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. 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.
27. 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`.
28. T. Menzies, S. Williams, O. El-waras, B. Boehm, and J. Hihn. How to avoid drastic software process change (using stochastic statbility). In *ICSE'09 (to appear)*, 2009. Available from `http://menzies.us/pdf/08drastic.pdf`.
29. 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`.
30. 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.

31. R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995. (reprinted 1997,2000).
32. R. Park. The central equations of the price software cost model. In *4th COCOMO Users Group Meeting*, November 1988.
33. 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.
34. L. Putnam and W. Myers. *Measures for Excellence*. Yourdon Press Computing Series, 1992.
35. L. Rela. Evolutionary computing in search-based software engineering. Master's thesis, Lappeenranta University of Technology, 2004.
36. 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`.

Appendix: Inside SEESAW

Recall our formalization of the model-based management task from the introduction: find the smallest subset of the project options $P' \subseteq P$ that most improves model estimates. This section describes the SEESAW algorithm and its methods for reducing the model estimates (as computed by Equation 5).

SEESAW was designed after observing experimentally that the most interesting part of the project options P was generally the minimum and maximum values for each variable. For example, if the project options of Figure 2 say that analyst capability ranges from low to *very high*, we kept finding that best effects were seen at either low or *very high*, and nowhere in between.

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 [27], we found that seesawing between $\{Low, High\}$ values was adequate for our purposes.

SEESAW is an adaption of Kautz & Selman's MaxWalkSat local search procedure [22]. Its pseudocode is presented in Figure 8. Each solution is scored via a Monte Carlo procedure (see `score` in Figure 8), and SEESAW seeks to *minimize* that score.

[Fig. 8 about here.]

SEESAW first combines the ranges for all the COCOMO project choices with the known project constraints of Figure 2. 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 20). 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 8 calls COCOMO / COQUALMO / THREAT models 100 times[7], each time selecting random values for the project choices (from the $Options$ set); and random values for the tuning options (as described in Figure 3). The median value of the Equation 5 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 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 [31]. 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 [15] 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

[7] The value "100" was set after experimentation showed that this was big enough to reduce our variance and not large enough to slow down the runtimes.

exploring uninformative areas of the problem. A stochastic search, on the other hand, does not get stuck in such uninformative areas.

List of Figures

		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. 1. The variables of COCOMO, COQUALMO, and the THREAT model.

	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. 2. Four case studies define a space of project options P.

increasing effort	decreasing effort
$0.073 \leq m \leq 0.21$	$-0.178 \leq m \leq -0.078$

Figure 3a: tuning options in COCOMO

phase	increasing defects	decreasing defects
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	$0 \leq m \leq 0.14$	$-0.19 \leq m \leq -0.053$

Figure 3b: tuning options in COQUALMO effort multipliers

phase	defect removal
requirements	$0.08 \leq m \leq 0.14$
design	$0.1 \leq m \leq 0.156$
coding	$0.11 \leq m \leq 0.176$

Figure 3c: tuning options in COQUALMO scale factors

Fig. 3. Linear relations in our models

	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. 4. An example risk table

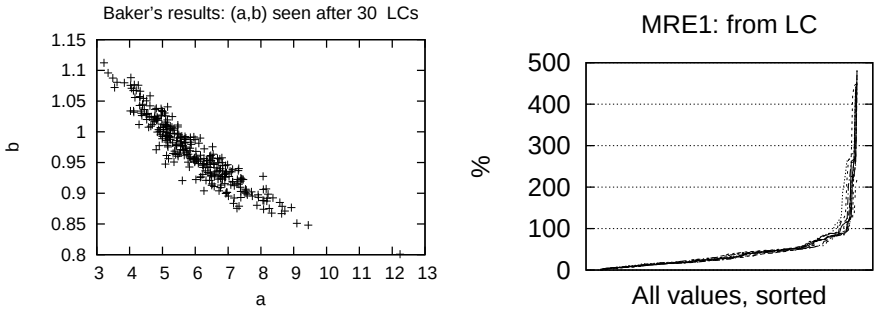

Fig. 5. Results of applying LC numerous times to 90% of the NASA93 data sets (available from `http://promisedata.org/data`). Left-hand-side shows computed (a, b) values. Right-hand-side shows MRE1s generated over the NASA93 data set for ten case studies (one study per line).

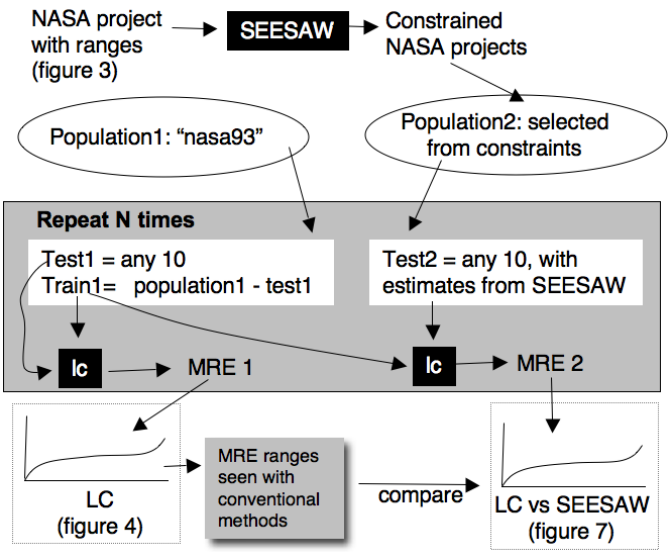

Fig. 6. Experimental design. "LC" denotes Boehm's 1981 regression procedure.

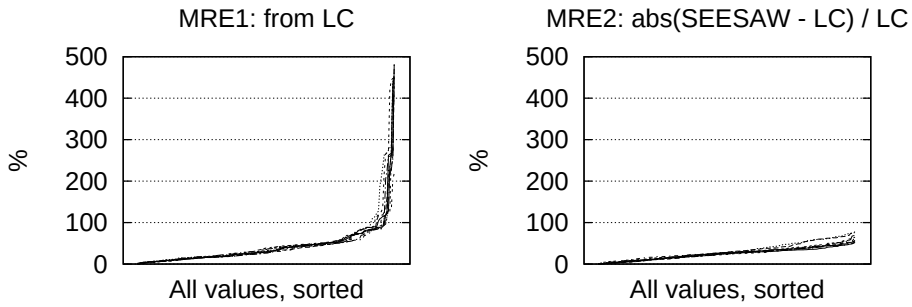

Fig. 7. MRE2 results for Figure 6, for ten case studies (one study per line). The MRE1 results (left-hand-side) come from Figure 4.

```
1  function run (AllRanges, ProjectConstraints) {
2    OutScore = -1
3    Prob = 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 Prob < rand()
16          then delete all ranges of X except Maybe from Out
17            delete X from Options
18            OutScore = MaybeScore
19       fi }
20   return backSelect(Out) }
21
22 function score(X, Value) {
23    Temp = copy(Out)  ;; don't mess up the Out global
24    from Temp, remove all ranges of X except Value
25    run monte carlo on Temp for 100 simulations, scoring each run using Equation 3.
26     return median score of the 100  }
```

Fig. 8. Pseudocode for SEESAW.