Accurate Estimates Without Calibration?

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

[1] LCSEE, West Virginia University, Morgantown, WV, USA, tim@menzies.us, oelrawas@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

Abstract. Most process models calibrate their internal settings using historical data. Collecting this data is expensive, tedious, and often an incomplete process. Is it possible to make accurate software process estimates without historical data? Suppose much of uncertainty in a model comes from a small subset of the model variables. If so, then after (a) ranking variables by their ability to constrain the output; and (b) applying a small number of the top-ranked variables; then it should be possible to (c) make stable predictions in the constrained space.

To test that hypothesis, we combined a simulated annealer (to generate random solutions) with a variable ranker. The results where quite dramatic: in one of the studies in this paper, we found process options that reduced the median and variance of the effort estimates by a factor of 20. In ten case studies, we show that the estimates generated in this manner are usually similar to those produced by standard local calibration.

Our conclusion is that while it is always preferable to tune models to local data, it is possible to learn process control options *without* that data.

1 Introduction

Without precise knowledge from an organization, it is difficult to make precise estimates about software processes at that site. For example, initial development effort estimates may be incorrect by a factor of four [7] or even more [17].

It can be very difficult to find relevant data within a single organization to fully specify all the internal parameters inside a process model. For example, after 26 years of trying, we have only collected less than 200 sample projects for the COCOMO database. There are many reasons for this, not the least being the business sensitivity associated with the data. Therefore, in this paper, we explore what can be decided from process models *without* local data.

* This research was conducted at WVU, USC, and NASA's Jet Propulsion Laboratory partially under a NASA sub-contract. 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.

For this experiment, we adopt the following framework. We say that a process model P yields estimates from a combination of $Project$ and $Model$ variables:

$$estimates = P(Project, Model)$$

P describes the space of influences between variables. and may take many forms:

– discrete-event models [16,19];
– system dynamics models [1];
– state-based models [3,13,23];
– rule-based programs [28];
– standard programming constructs such as those used in Little-JIL [11,33];
– or the linear models used in COCOMO [7,9], PRICE-S [29] and SEER-SEM [15].

The strength of each influence is controlled by the $Model$ variables. Taken together, the process model P and the $Model$ variables store what we've leaned from the *past*.

$Project$ variables, on the other had, concern a *new* situation that should be analyzed using past knowledge. For example, P could assert "$effort \propto pcap$" (programmer skills is proportional to development effort) while $Model$ could assert the proportionality constant of -0.7 (i.e. "$effort = -0.7pcap$"). Finally, $Project$ could assert that programmer skills are "in the upper range"; e.g. for a COCOMO model "$pcap \in \{4, 5\}$".

We say $Project$ and $Model$ variables can be:

– $fixed$ to one value such as "programmer capability ($pcap$) is nominal";
– $free$ to take on any legal value. In COCOMO, a free $pcap$ can take values

$$\{veryLow = 1, low = 2, nominal = 3, high = 4, veryHigh = 5\}$$

– or $float$ to some subset of the whole range. For example, a manager might declare that "our programmers are in the upper ranges"; i.e. this $pcap$ floats in a particular part of the entire $pcap$ range ($pcap \in \{4, 5\}$).

The range of legal values for variables increases from $fixed$ to $float$ to $free$:

$$(|fixed| = 1) < |float| < |free|$$

This paper reports an experiment that frees *both* the $Model$ and $Project$ variables. At first glance, such an experiment may seem perverse, particularly if the goal is to reduce uncertainty. Free variables range over a larger space than fixed variables: the more free variables, the wider the range of $Estimates$. If we free both $Model$ and $Project$ variables then, surely, this will result in greater $Estimate$ uncertainty?

However, our analysis is not just some passive observer of a large space of options. Instead, it is an active agent that seeks parts of the options space where predictions can be made with greater certainty. We augment a Monte Carlo analysis with two tools. SA is a *simulated annealing algorithm* that minimizes $Estimates$. RANKER is a *variable pruning algorithm*, that seeks the *smallest* number of $Project$ variables that *most* reduce the $Estimates$. The combination of SA+RANKER is called STAR[4].Since it

	Monte Carlo	STAR
case study	SCAT	STAR
flight	712	44
ground	389	18
OSP	629	68
OSP2	84	31

Fig 1a: variance, in months.

	Monte Carlo	STAR
case study	SCAT	STAR
flight	1357	86
ground	737	38
OSP	1951	410
OSP2	297	182

Fig 1b: median, in months.

Fig. 1. Effort estimates seen in 1000 simulations of the $Project$ ranges found by STAR. "Variance" (left hand side) shows the difference between the 75th and 50th percentile. "Median" (right hand side) shows the 50th percentile estimate.

knows the most influential $Project$ ranges, STAR can discover (and then constrain) the factors that most most impact $Estimates$.

When compared to state-of-the-art process models, the effects of a STAR-style analysis are quite dramatic. Figure 1 compares STAR's estimates to those generated by SCAT [20–22], a COCOMO-based tool used at NASA's Jet Propulsion Laboratory. SCAT fixes $Model$ and perform a Monte Carlo simulation of the $Project$ ranges. Each row of Figure 1.A is one case study:

- $flight$ and $ground$ systems software from NASA's Jet Propulsion Laboratory;
- OSP is the GNC[5] for NASA's Orbital Space Plane (prototype);
- $OSP2$ is a newer version of OSP.

Note that, for all four case studies, STAR reduces the variance and median estimates to a small fraction of SCAT's estimates, sometimes as much as a factor of 20 (in Figure 1a: $\frac{712}{44} \approx 20$; in Figure 1b: $\frac{737}{38} \approx 20$).

The rest of this paper describes STAR. We extend prior work in two ways. Prior reports on STAR [25] were based on limited case studies; here we report ten new case studies showing that our main effect (reduced median and variance) holds in a wide range of cases. Also, prior reports on Figure 1 [24] failed to check the validity of those results. The ten case studies discussed below show that STAR's estimated are shown to be close to those generated via standard local calibration, despite being generated from a large space of $Project$ and $Model$ options. This validity check greatly increases our confidence in the STAR method.

It is unknown if our results apply to software process models more complex than STAR's COCOMO-style of models. However, our results to date suggest that other process models could make reasonably accurate predictions without local data by:

- finding the fewest number of variables that most effect model output;
- constrain them;
- check for stable conclusions in the constrained space.

[4] The name is a geek joke. In regular expressions, the star meta-character "*" matches any characters. That is, just like STAR, it can be used to search a wide range of options.

[5] GNC= guidance, navigation, and control

2 Related Work

In terms of the framework of this paper, related work may be divided into:

– *Prediction*: fix $Model$ and $Project$ and generates fixed $estimates$.
– *Calibration*: import an log of fixed $estimates$ and $Project$ variables, find fixes to $Model$ that best explain how $Project$ inputs lead to $estimation$ outputs.
– *Monte Carlo* studies: fix the $Model$ values (perhaps to values learned via calibration), import floating $Project$ values, generates a range of possible $estimates$.

In the field of effort estimation:

– *Prediction* is used to create one point estimate for a project; e.g. COCOMO [7, 8],PRICE-S [29] and SEER-SEM [15].
– *Calibration* is useful for learning from historical data; e.g. see Boehm's local calibration procedure [7, p526-529] or the COSEEKMO toolkit [26].
– *Monte Carlo* studies are useful for conducting what-if queries across a range of possible projects [30]. Such Monte Carlo studies are conducted by many tools including COBRA [10], CrystalBall [5], SCAT [21, 22], and 2CEE [6].

To the best of our knowledge, this work is the first to try freeing *both* the $Project$ and $Model$ variables. 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-liner search methods like SA. A recent review of 123 search-based SE papers [31] 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 facets that distinguished STAR from other methods is that we are searching over more than just the effort models explored by the Aquilar-Ruiz & Alvarez teams. Also, unlike standard data mining approach, we do not try to learn better $Model$ variables from historical data.

3 STAR

STAR's current implementation explores three software process models:

– The COQUALMO software defect predictor [9, p254-268].
– The COCOMO software effort predictor [9, p29-57].
– The THREAT predictor for project effort & schedule overrun [9, 284-291].

COQUALMO models two processes (defect introduction and defect removal) for three phases (requirements, design, coding). COCOMO assumes that effort is exponentially proportional to some *scale factors* and linearly proportional to some *effort multipliers*. COCOMO estimates are development months (225 hours) and includes all coding, debugging, and management activities. The THREAT model contains a large set of two-dimensional tables representing pairs of variable settings 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).

STAR samples the space of possibles models inside COCOMO and COQUALMO using the following technique. Internally, COCOMO and COQUALMO models contain many linear relationships. Nominal values of $x = 3$ change some estimate by a factor of one. These COCOMO lines can hence be modeled as a straight line $y = mx + b$ passing through the point $x, y = 3, 1$. Such a line has a y-intercept of $b = 1 - 3m$. Substituting this value of b into $y = mx + b$ yields $y = m(x - 3) + 1$. COCOMO's effort slopes are either positive or negative, denoted m^+, m^- (respectively):

- The positive slopes m^+ represents the variables that are proportional to effort; e.g. $increasing$ required reliability also $increases$ the development effort.
- The negative slopes m^- represents the variables that are *inversely* proportional to effort; e.g. $increasing$ analyst capability $decreases$ the development effort.

Based on decades of experiments with calibrating COCOMO models, we have identified variables with different slopes. These following COCOMO variables have m^+ slopes: cplx, data, docu, pvol, rely, ruse, stor, and time. Also, these variables have m^- slopes acap, apex, ltex, pcap, pcon, plex, sced, and site (for an explanation of those terms, see Figure 2). Further, based on decades of calibration of COCOMO models, we assert that effort estimation, m^+ and m^- have the ranges:

$$\begin{array}{c} -0.178 \leq m^- \leq -0.078 \\ 0.073 \leq m^+ \leq 0.21 \end{array} \tag{1}$$

Using an analogous procedure, it is possible to derive similar equations for the COCOMO scale factors, the COQUALMO scale factors/effort multipliers/ defect removal variables (for full details, see [25]).

With the above machinery, it is now possible to define a Monte Carlo procedure to sample the space of possible THREAT/COCOMO/COQUALMO $Models$: just randomly selecting $\{m^-, m^+\}$. As to sampling the space of possible THREAT models, this is achieved by adding random variables to the cells of THREAT's tables.

STAR tries to minimize defects (D), threats (T), and development effort (E). This is a non-linear optimization function: e.g. reducing costs can introduce more defects. For this reason, we use simulated annealing (SA) to explore trade-offs between models. SA is best explained in comparison to the Metropolis algorithm.

A *Metropolis* Monte Carlo algorithm [27] improves on basic Monte Carlo as follows. New solutions are created by small mutations to some *current* solutions. In the case of STAR, an "solution" is some randomly selected part of the space of possible $Projects$. If a new solution is "better" (as assessed via an *energy function*), it becomes the new *current* solution used for future mutations. STAR's energy function is $E = \sqrt{\overline{E}^2 + \overline{D}^2 + \overline{T}^2} / \sqrt{3}$ where $\overline{x}$ is a normalized value $0 \leq \frac{x - min(x)}{max(x) - min(x)} \leq 1$. Energy ranges $0 \leq E \leq 1$ and *lower* energies are *better*. If a new solution does not have lower energy, a Boltzmann acceptance criteria is used to probabilistically decide to assess the new state: the worse the new state, the less likely that it becomes the new current state.

A *simulated annealer* (SA) [18] adds a "temperature" variable to the Boltzmann accept criteria such that, at high temperatures, it is more likely that the algorithm will

		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.

jump to a new worst current state. This allows the algorithm to jump out of local minima while sampling the space of options. As the temperature cools, such jumps become less likely and the algorithm reverts to a simple hill climber.

Our *RANKER* algorithm instruments the internals of SA. Whenever a solution is assigned some energy, that energy is added to a counter maintained for each variable setting in $Projects$. When SA terminates, RANKER sorts all variable ranges by the sum of the energies seen during their use. The ranges that are lower in the sort order are associated with lower energy solutions; i.e. lower defects, efforts, threats. RANKER then conducts experiments where it fixes the first N ranked ranges and lets the remaining variables float. N is increased till some minimum energy point is reached. A *policy* are the project settings that achieve that minimum energy point.

The last two columns of Figure 2 show the results of Delphi panel session at JPL where the COCOMO variables were separated into those $tactical$ variables that can be changed within the space of one project, and those $strategic$ variables that required 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. In the sequel, all our RANKER experiments will be divided into those that just use the $strategic$ variables and those that just use the $tactical$ variables[6].

[6] Note that these definitions of $strategic$ and $tactical$ choices are not hard-wired into STAR. If a user disagrees with our definitions of strategic/tactical, they can change a simple configuration file.

	float			fixed	
project	variable	low	high	variable	setting
	prec	1	2	data	3
OSP	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		
	prec	3	5	flex	3
OSP2	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
	rely	3	5	tool	2
	data	2	3	sced	3
flight	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		
	rely	1	4	tool	2
	data	2	3	sced	3
ground	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.

4 Experiments

Figure 3 shows various $Projects$ expressed in term of $float$ing and $fixed$ variables. For example, with JPL's flight systems, the $rely$ (required reliability) can float anywhere in the upper range; i.e. $rely \in \{3, 4, 5\}$. However, for flight systems, $sced$ (schedule pressure) is tightly defined (so $sced$ is fixed to the value 3).

Figure 4 and Figure 5 shows the results of STAR. The variable ranges are sorted along the x-axis according the order generated by RANKER. At any x value we see the results of fixing the ranges $1..x$, letting all ranges $x + 1...max$ float, then running 1000 Monte Carlo simulations. In the results, "median" refers to the 50th percentile band and "spread" refers to the difference between the 75th and 50th percentile in the 1000 generate estimates.

For this paper, we ran SA+RANKER on the four case studies of Figure 3, plus a fifth study called "ALL"" that used the entire COCOMO ranges, unconstrained by a particular project. Each study was repeated twice- one for controlling just the strategic variables and once for controlling just the tactical variables. This resulted in ten experiments.

X	variable = setting
1	pmat = 4
2	ltex = 4
3	acap = 3
4	apex = 3
5	prec = 2
6	pcon = 3
7	execution testing and tools = 6
8	peer reviews = 6
9	automated analysis = 6

Fig.4.A: controlling only strategic $Project$ variables

X	variable = setting
1	ruse = 2
2	cplx = 5
3	resl = 3
4	tool = 3
5	sced = 2
6	stor = 3
7	flex = 5
8	automated analysis = 6
9	peer reviews = 6
10	docu = 2
11	execution testing and tools = 6
12	sced = 1

Fig.4.B: controlling only tactical $Project$ variables

Fig. 4. Some RANKER results on OSP. The settings shown under the plots describe the policy that leads to the policy point.

X	feature = range
1	pmat = 3
2	site = 6
3	pcon = 5
4	plex = 4
5	pcap = 5
6	ltex = 4
7	apex = 5
8	prec = 5
9	acap = 5
10	automated analysis = 6
11	execution testing and tools = 6
12	peer reviews = 6
13	acap = 4

Fig.5.A: controlling only strategic $Project$ variables

X	feature = range
1	resl = 5
2	cplx = 1
3	execution testing and tools = 6
4	flex = 5
5	docu = 1
6	ruse = 2
7	data = 3

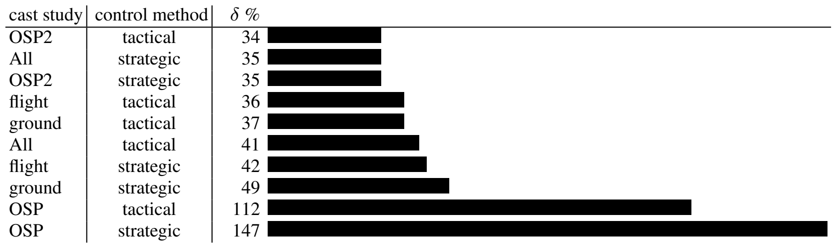
Fig.5.B: controlling only tactical $Project$ variables.

Fig. 5. Some RANKER results on JPL ground systems. The settings shown under the plots. describe the policy that leads to the policy point.

Some of the results from four of those experiments are shown in Figure 4 and Figure 5 (space restrictions prevent us from showing all the results). In those four experiments (and in the other six, not shown) the same effect was observed. Minimum effort and defects was achieved after fixing a small number of $Project$ variables (in Figure 4.A, Figure 4.B, Figure 5.A, and Figure 5.B, that number was at X={9,12,13 7} respectively). At these minimum points, the median and spread estimates were greatly reduced. We call this minimum the $policypoint$ and use the term $policy$ to refer to the intersection of the case study defined in Figure 3, and the ranges found in the range between $\{1 \leq x \leq policypoint\}$.

Figure 4 and Figure 5 are the reports we would offer back to the manager. Start at the top of this list, we would advise, and apply as many oft eh top N things that you can. Do not waste time implementing policy changes off this list.

In terms of controlling uncertainty, the reduction in the spread estimates at the policy point is particularly interesting. Note that this reduction in model uncertainty was achieved by only controlling a few of the $Project$ variables while letting all other $Project$ and $Model$ variables float free. That is, in these case studies, projects could



cast study	control method	δ %
OSP2	tactical	34
All	strategic	35
OSP2	strategic	35
flight	tactical	36
ground	tactical	37
All	tactical	41
flight	strategic	42
ground	strategic	49
OSP	tactical	112
OSP	strategic	147

Fig. 6. Median $\delta = (estimate(STAR) - estimate(lc))$ between effort estimates generated by conventional means (LC) and STAR.

be controlled (development effort and defects reduced) without using historical data to constrain the $Model$ variables.

For each of our ten experiments, a set of random $Projects$ were generated, consistent with the policies; i.e.

- If the policy fixes a value, then the $Project$ contains that value;
- Otherwise, if the variable is found Figure 3, it is drawn from those constraints;
- Otherwise, the variable's value is selected at random from background knowledge of the legal range of the Figure 2 variables.

For each set, the following procedure was repeated 20 times. Ten examples were removed at random and Boehm's local calibration (LC) procedure [7, p526-529] was used to train a COCOMO model on the remaining $Project$ examples[7]. LC's estimates were then compared to the estimates generated by STAR's simulation at the policy point (i.e. floating over both the policy and the $Model$ ranges). Figure 6 show the median difference in the estimates generated by LC or STAR . Note that, in $\frac{8}{10}$ cases, the difference is under 50%. The reason for the large deltas seen in $\frac{2}{10}$ of the results (from the OSP case study) are currently unknown but are a subject of much current exploration.

The median δ values of Figure 6 are around 0.4; i.e. a STAR estimate of 100 months could really range for 60 to 140 months. Compared to the effort estimate reductions shown in the introduction, δ is quite small. Recall that STAR reduced effort estimates to a small part of the initial values, sometimes a factor of 20; i.e by a factor that is much larger than 0.4. Clearly, even if STAR is wrong by $\pm 40\%$, then the overall benefits to be gained from applying STAR's policies are still dramatically large.

5 Discussion

Given all the randomized exploration STAR performs over the space of possible $Models$, this discrepancy is very small. and those discrepancies are dwarfed by the much larger effort reductions of Figure 1.

[7] LC was chosen since, in extensive experiments, we have found this decades old procedure to be remarkably competitive with current data mining methods [14] including bagging and boosting [6].

How are we to explain the remarkable effectiveness of STAR in managing uncertainty? Researchers in planning and theorem proving have recently shown that as model complexity grows, other constraining effects may appear such as "master variables"; i.e. a small number of settings that control all other settings [12, 32]. Such master variables can greatly reduce the search space within large models.

We hypothesize that software process models also contain master variables; i.e. much much of uncertainty in a model is due to the influence of a small subset of model variables. If so, then after (a) ranking variables by their ability to constrain the output; and (b) applying a small number of the top-ranked variables; then it should be possible to (c) make stable predictions in the constrained space.

6 Conclusion

In studies with one widely-used suite of effort/ detect/ threat predictors for software systems, we have shown that:

- $Estimation$ median values can be greatly reduced (see Figure 1). In comparisons with other effort estimation tools, the reduction can quite dramatic. In the best case our tools found $Project$ ranges that yields estimates that were 5% of estimates found by other means.
- $Estimation$ variance can be reduced by only floating the $Project$ values and leaving the $Model$ values free (see Figure 4 and Figure 5).
- Within the space of $Project$ options that most reduce $Estimation$ median and variance, the predictions made by our process models are remarkably similar to those made by conventional methods (see Figure 6).

The first result suggests that it may be highly advantageous to use STAR. Projects designed around STAR's recommendations will be will be delivered sooner and have fewer bugs or threats.

The second result is of much practical importance since it means we do not require calibration data to tune the $Model$ variables. If process models can be deployed without calibration, then they can be used with much greater ease and *without* the requirement for an expensive and time-consuming period of data collection.

The third result is showing that (a) this method can find and remove the major sources of uncertainty in a project; (b) in the reduced space, it is possible that the estimates in the resulting constrained space will be close to estimates found via tuning on historical data. In the above discussion section, we commented that this result has precedent in the AI planning and theorem proving literature.

Finally, we comment on the external validity of these results. Compared to many other process models[8] this combination of effort/threat/defect models is relatively simple. As model complexity grows, then the space of possible $Estimates$ can grow exponentially and STAR's controlling effect may disappear. Therefore it is clear that we can not claim that, for *all* process models, that $Estimate$ variance can be controlled by just constraining $Project$, not $Model$, variance.

[8] See Software Process journal, issue on Software Process Simulation, vol. 7, No. 3-4, 2002.

Nevertheless, data collection for the purposes of model calibration remains as a expensive, tedious, and often incomplete process. Our results suggest that such data collection may be, for some process models, an optional activity (caveat: provided that a process model exists that specifics the general relationships between concepts in a domain). Our hope is that the results of this paper encouraging enough that other software process modeling researchers will try the following strategy:

– finding the fewest number of variables that most effect model output;
– constrain them;
– check for stable conclusions in the constrained space.

If these results from STAR generalize to more complex models, then is should be possible to make reasonably accurate predictions without local calibration data.

Note that if such stability is *absent* in more complex models, and those models are being used in domains with data collection problems, then we would argue that that is a reason to abstain from such complexity, and use COCOMO-style models instead.

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. John Bailey. Using monte carlo and cocomo-2 to model a large it system development, 2002.
6. 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`.
7. B. Boehm. *Software Engineering Economics*. Prentice Hall, 1981.
8. 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`.
9. 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.
10. 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.
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. J. Crawford and A. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *AAAI '94*, 1994.

13. D. Harel. Statemate: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16(4):403–414, April 1990.
14. Omid Jalali. Evaluation bias in effort estimation. Master's thesis, Lane Department of Computer Science and Electrical Engineering, West Virginia University, 2007.
15. R. Jensen. An improved macrolevel software development resource estimation model. In *5th ISPA Conference*, pages 88–92, April 1983.
16. D. Kelton, R. Sadowski, and D. Sadowski. *Simulation with Arena, second edition*. McGraw-Hill, 2002.
17. C.F. Kemerer. An empirical validation of software cost estimation models. *Communications of the ACM*, 30(5):416–429, May 1987.
18. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.
19. A. Law and B. Kelton. *Simulation Modeling and Analysis*. McGraw Hill, 2000.
20. K. Lum, J. Powell, and J. Hihn. Validation of spacecraft cost estimation models for flight and ground systems. In *ISPA Conference Proceedings, Software Modeling Track*, May 2002.
21. Karen Lum. Software cost analysis tool user document, 2005.
22. Karen Lum, Michael Bramble, Jairus Hihn, John Hackney, Mori Khorrami, and Erik Monson. Handbook for software cost estimation, 2003.
23. 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.
24. 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`.
25. 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`.
26. 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`.
27. N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. *J. Chem. Phys*, 21:1087–1092, 1953.
28. 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.
29. R. Park. The central equations of the price software cost model. In *4th COCOMO Users Group Meeting*, November 1988.
30. 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.
31. L. Rela. Evolutionary computing in search-based software engineering. Master's thesis, Lappeenranta University of Technology, 2004.
32. R. Williams, C.P. Gomes, and B. Selman. Backdoors to typical case complexity. In *Proceedings of IJCAI 2003*, 2003. `http://www.cs.cornell.edu/gomes/FILES/backdoors.pdf`.
33. 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`.