How to Avoid Drastic Software Process Change (using Stochastic Stability)

Tim Menzies,* Steve Williams,
Oussama El-Rawas
CSEE, WVU, USA
tim@menzies.us
swilli12@mix.wvu.edu
oelrawas@mix.wvu.edu

Barry Boehm
Computer Science Dept.
Uni. S. California, USA
boehm@sunset.usc.edu

Jairus Hihn
Jet Propulsion Laboratory,
California Inst. of Technology,
California, USA
jairus.hihn@jpl.nasa.gov

Abstract

Before performing drastic changes *to a project, it is worthwhile to thoroughly explore the available options within the current structure of a project. An alternative to drastic change are* internal changes *that adjust current options within a software project. In this paper, we show that the effects of numerous internal changes can out-weigh the effects of drastic changes. That is, the benefits of drastic change can often be achieved without disrupting a project.*

The key to our technique is SEESAW, a novel stochastic stability tool that (a) considers a very large set of minor changes using stochastic sampling; and (b) carefully selects the right combination of effective minor changes.

Our results show, using SEESAW, project managers have more project improvement options than they currently realize. This result should be welcome news to managers struggling to maintain control and continuity over their project in the face of multiple demands for drastic change.

1. Introduction

Software process control is complicated by uncertainty. Exactly how good are our analysts? How large is the final system? Do we understand our tools? These are some of the questions that most managers cannot answer precisely.

For example, if asked "what is the required reliability on this project?" a manager may reply "well, the code has to run but this is not a safety critical system". In terms of, say, the COCOMO [6] five-point reliability scale, the manager's answer describes a *range of options* "low" to "high". Some of options are *controllable*; e.g. the manager may offer the customer an early delivery of the software if the customer considers *not* building a "high" reliability system.

This paper compares the effects of (a) *internal changes* to a project that adjust the controllables within their current known ranges and (b) *drastic changes* that require moving a project outside of current bounds. For example, for projects with a range of reliability options "low" to "high", an *internal change* would be setting reliability to a value within "low" to "high". On the other hand, a *drastic change* would be to suddenly demand that a project becomes a "very high" reliability project. Figure 1 lists the disruptive changes of nine drastic changes. For example, "improve personnel" (by firing staff and hiring new programmers) may lead to a major union dispute. Other drastic changes like "reduce functionality" can restrict the functionality of the next release. This, in turn, could mean less revenue if the reduced functionality software has less market appeal.

SEESAW is a tool that seeks alternatives to drastic change. SEESAW stochastically explores combinations of internal changes. When controlling mission-critical software, such stochastic methods are not desirable. However, when studying the uncertainty associated with unknowns

Drastic change	Possible undesirable impact
1 Improve personnel	Firing and re-hiring personnel leading to wide-spread union unrest.
2 Improve tools, techniques, or development platform	Changing operating systems, IDEs, coding languages
3 Improve precedentness / development flexibility	Changing the goals of the project and the development method.
4 Increase architectural analysis / risk resolution	Far more elaborate early life cycle analysis.
5 Relax schedule	Delivering the system later.
6 Improve process maturity	May be expensive in the short term.
7 Reduce functionality	Delivering less than expected.
8 Improve the team	Requires effort on team building.
9 Reduce quality	Less user approval, smaller market.

Figure 1. Nine drastic changes from [3].

*Part of the research described in this paper 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.

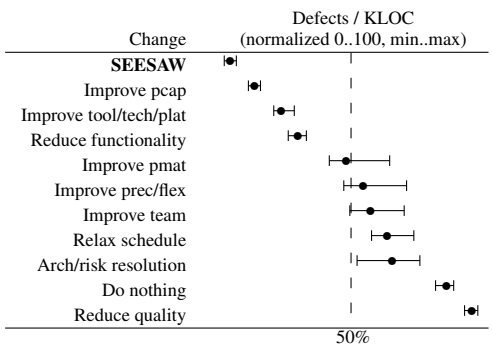

Figure 2. Defect predictions from changing a project (25 to 75% percentile range from 100 simulations; median values shown as a dot).

about software processes, stochastic algorithms like SEESAW are very useful. If run multiple times, stochastic sampling can return multiple options so users can study the range of possible solutions. That is, SEESAW offers both a solution and a comment on the *stability* of that solution.

This paper checks the stability of predictions about drastic change or internal changes learned by SEESAW's stochastic sampling. For the drastic changes of Figure 1 and four NASA case studies, our main result is that:

> *Combinations of internal changes usually perform as well or better than drastic changes.*

That is, using internal changes, the benefits of drastic change can be achieved without project disruption. For example, Figure 2 compares defect/KLOC predictions from SEESAW and the drastic changes of Figure 1. SEESAW's recommendations, comprising only internal changes, resulted in the smallest defect predictions (see line 1).

The rest of this paper explains how Figure 2 was generated. After some preliminaries, we discuss the options available within the current structure of four sample projects. Figure 1's drastic changes will then be implemented as operators on those projects. Next we describe SEESAW and the models it operates on: the COCOMO effort and time estimator [7, p29-57]; and the COQUALMO defect predictor [7, p254-268]. In the results that follow we show that, for our case studies, SEESAW's search through the space of internal changes usually out-performs the drastic changes of Figure 1.

The contribution of this paper is a demonstration that managers have more options than they may realize. Stochastic stability is an insightful method for discovering useful project reconfigurations that improve a project while avoiding requiring drastic and disruptive project change.

2. Preliminaries

The reader may wonder why we use a stochastic method like SEESAW to explore project options. Would not a simpler method suffice? For example, in the case of *linear models* that have been *precisely tuned* using local data, it is a simple matter to check if a combination of internal changes does better than drastic change. Many of the relationships inside COCOMO model are linear. For such models, "what-if" queries require just a simple *linear extrapolation* to assess the relative effectiveness of, say, reducing functionality versus some combination of internal changes.

Unfortunately, not all tunings are *precise*. Sometimes, even after tuning, the gradient of the relationships may not be known with certainty. For example, the COCOMO effort model predictions are affected linearly and exponentially by two features a, b. Baker [2] tuned these a, b values using data from NASA systems. After thirty 90% random samples of that data, the a, b ranges were surprisingly large:

$$(2.2 \leq a \leq 9.18) \wedge (0.88 \leq b \leq 1.09) \qquad (1)$$

Elsewhere we have been partially successful in reducing the range of Equation 1 with feature subset selection (FSS) [8, 14] or more data collection. FSS reduces but does not eliminate the a, b variance. Also, further data collection is possible, but only at great organizational expense. This is due to 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 example, after two years we were only able to add 7 records to a NASA-wide software cost metrics repository [16]. Having failed to generate precise tunings, we developed SEESAW to discover what stable conclusions we could find within the space of possible tunings.

Another drawback with simplistic linear extrapolation is that, when optimizing for effort *and* time *and* defects, there may be contradictory effects. For example, we show below one result where effort *reduced*, but defects *increased dramatically*. Hence, optimizing our models is not a simple matter of moving fixed distances over some linear effect: there are also some trade-offs to be considered (e.g. using a tool that considers combinations of effects, like SEESAW).

3. Options Within Projects

SEESAW ranks project changes using the COCOMO and COQUALMO effort, time, and defect predictors. The terminology of those models is reviewed in Figure 3.

Using that terminology, Figure 4 summarizes four NASA case studies:

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

	Definition	Low-end = {1,2}	Medium ={3,4}	High-end= {5,6}
Defect removal features				
execution-based testing and tools (etat)	all procedures and tools used for testing	none	basic testing at unit/ integration/ systems level; basic test data management	advanced test oracles, assertion checking, model-based testing
automated analysis (aa)	e.g. code analyzers, consistency and traceability checkers, etc	syntax checking with compiler	Compiler extensions for static code analysis, Basic requirements and design consistency, traceability checking.	formalized specification and verification, model checking, symbolic execution, pre/post condition checks
peer reviews (pr)	all peer group review activities	none	well-defined sequence of preparation, informal assignment of reviewer roles, minimal follow-up	formal roles plus extensive review checklists/ root cause analysis, continual reviews, statistical process control, user involvement integrated with life cycle
Scale factors:				
flex	development flexibility	development process rigorously defined	some guidelines, which can be relaxed	only general goals defined
pmat	process maturity	CMM level 1	CMM level 3	CMM level 5
prec	precedentedness	we have never built this kind of software before	somewhat new	thoroughly familiar
resl	architecture or risk resolution	few interfaces defined or few risks eliminated	most interfaces defined or most risks eliminated	all interfaces defined or all risks eliminated
team	team cohesion	very difficult interactions	basically co-operative	seamless interactions
Effort multipliers				
acap	analyst capability	worst 35%	35% - 90%	best 10%
aexp	applications experience	2 months	1 year	6 years
cplx	product complexity	e.g. simple read/write statements	e.g. use of simple interface widgets	e.g. performance-critical embedded systems
data	database size (DB bytes/SLOC)	10	100	1000
docu	documentation	many life-cycle phases not documented		extensive reporting for each life-cycle phase
ltex	language and tool-set experience	2 months	1 year	6 years
pcap	programmer capability	worst 15%	55%	best 10%
pcon	personnel continuity (% turnover per year)	48%	12%	3%
plex	platform experience	2 months	1 year	6 years
pvol	platform volatility ($\frac{frequency\ of\ major\ changes}{frequency\ of\ minor\ changes}$)	$\frac{12\ months}{1\ month}$	$\frac{6\ months}{2\ weeks}$	$\frac{2\ weeks}{2\ days}$
rely	required reliability	errors are slight inconvenience	errors are easily recoverable	errors can risk human life
ruse	required reuse	none	multiple program	multiple product lines
sced	dictated development schedule	deadlines moved to 75% of the original estimate	no change	deadlines moved back to 160% of original estimate
site	multi-site development	some contact: phone, mail	some email	interactive multi-media
stor	required % of available RAM	N/A	50%	95%
time	required % of available CPU	N/A	50%	95%
tool	use of software tools	edit,code,debug		integrated with life cycle

Figure 3. Features of the COCOMO and COQUALMO models used in this study.

- OSP2 is a later version of OSP;
- Flight and ground systems reflect typical ranges seen at NASA's Jet Propulsion Laboratory.

Some of the features in Figure 4 are known precisely (see all the features with single *values*). But many of the features in Figure 4 do not have precise values (see all the features that *range* from some *low* to *high* value). Sometimes the ranges 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 4 does not mention all the features listed in Figure 3 inputs. For example, our defect predictor has inputs for use of *automated analysis*, *peer reviews*, and *execution-based testing tools*. For all inputs not mentioned in Figure 4, values are picked at random from the full range of Figure 3.

4. Imposing Drastic Changes

Using Figure 4, we can now clearly distinguish between *drastic* and *internal* changes. Drastic changes means reorganizing a project *outside* of the ranges shown in Figure 4, while internal changes reorganizes a project *within* those ranges. Internal changes merely move the project into some subset of the current project assumptions. Drastic changes, on the other hand, can completely alter those assumptions. Take as an example the "improve process maturity" drastic

	ranges			values	
project	feature	low	high	feature	setting
OSP: Orbital space plane	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
JPL flight software	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		
JPL ground software	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		

Figure 4. Four case studies. Numeric values $\{1, 2, 3, 4, 5, 6\}$ map to $\{verylow, low, nominal, high, veryhigh, extrahigh\}$.

change from Figure 1. According to Figure 4, JPL ground systems have process maturities in the range 2 to 3 (low to nominal). Demanding very high CMM levels for these JPL ground systems would likely take years, require the commitment of very senior JPL management, and necessitate extensive retraining of local staff and/or hiring of new staff.

Figure 5 defines the values we imposed on each case study as part of each drastic change. Most of the values in Figure 5 are self-explanatory with two exceptions.

Drastic change	**Effects on Figure 4**
1 Improve personnel	acap = 5; pcap = 5; pcon = 5 apex = 5 ; plex = 5 ; ltex = 5
2 Improve tools, techniques, or development platform	time = 3; stor = 3 pvol = 2; tool = 5 site = 6
3 Improve precedentness / development flexibility	prec = 5; flex = 5
4 Increase architectural analysis / risk resolution	resl = 5
5 Relax schedule	sced = 5
6 Improve process maturity	pmat = 5
7 Reduce functionality	data = 2; kloc * 0.5
8 Improve the team	team = 5
9 Reduce quality	rely = 1 ; docu = 1 time = 3 ; cplx = 1

Figure 5. Implementing drastic changes.

Firstly, the $kloc * 0.5$ in "reduce functionality" means that, when imposing this drastic change, we only implement half the system. Secondly, most of the features fall in the range one to five. However, some have minimum values of 2 or higher (e.g., $pvol$ in "improve tools/tech/dev"), and some have maximum values of 6 (e.g., $site$ in "improve tools/tech/dev"). This explains why some of the drastic changes result in values other than one or five.

To impose a drastic change on a case study, if that change refers to feature X (in the right-hand column of Figure 5), then we first (a) removed X from the values and ranges of the case study (if it was present); then (b) added the changes of Figure 5 as fixed values for that case study.

5. Finding Alternatives to Drastic Change

SEESAW searches within the ranges of Figure 4 to find constraints that most reduce development effort, development time, and defects. Figure 6 shows SEESAW's pseudo-code. The code is an adaption of Kautz & Selman's MaxWalkSat local search procedure [13]. The main changes are that each solution is scored via a Monte Carlo procedure (see `score` in Figure 6) and that SEESAW seeks to *minimize* that score (since, for our models it is some combination of defects, development effort, and development time). Other changes are discussed below.

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

In the case where a feature is $fixed$ to a single value, then $Low = High$. Since there is no choice to be made for this feature, SEESAW ignores it. The algorithm ex-

plores only those features 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 feature X will be discovered. If so, the range for X is reduced to that single value, and the feature is not examined again (line 17).

SEESAW prunes the final recommendations (line 21). This function pop off 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 6 calls COCOMO/COQUALMO models 100 times, each time selecting random values for each feature $Options$. The median value of these 100 simulations is the $score$ for the current project settings. 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 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 the algorithm seesaws 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 [16], we found that seesawing between $\{Low, High\}$ values was adequate for our purposes.

SEESAW is a stochastic algorithm: the selection of the next feature 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 [17]. For example, we have implemented a deterministic version of SEESAW that replaces the random selection of *one* feature in line 7 with a search through *all* features 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 [9] 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.

SEESAW incrementally grows solutions from unconstrained (where all features can take any value in $\{Low, High\}$) to fully constrained (where all features are set to a single value). This is unlike simulated annealing or MaxWalkSat, which simultaneously offer settings to all features at every step of their reasoning. Figure 7 shows a

```
 1 function run (AllRanges, ProjectConstraints) {
 2   OutScore = -1
 3   P = 0.95
 4   Out = combine(AllRanges, ProjectConstraints)
 5   Options = all Out features 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
27   return median score from monte carlo simulations
28 }
```

Figure 6. Pseudocode for SEESAW.

single run of SEESAW: each dot marks one selection (lines 16,17,18 of Figure 6). In our terminology, selecting one value from a range is a *decision* on all values in the range. For example, if Figure 4 includes a process maturity of (3,4,5) and SEESAW selects "5", then that is three decisions. (this is why the 29 dots of Figure 7 result in 100 decisions). Note how, as decisions are made, the score is minimized. Score minimization is desirable since our scores are calculated from a combination of project predictions that we want to reduce (total effort, defects, development time).

Incremental decision making is an important property of SEESAW. Observe how, in Figure 7, 50% of the score

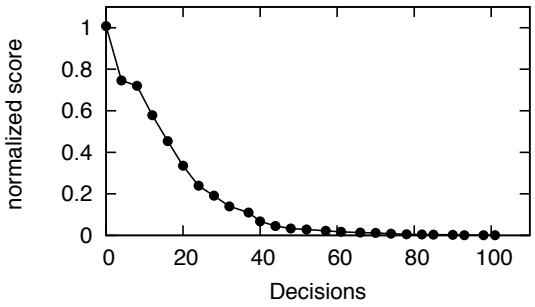

Figure 7. Single run of SEESAW, score normalized min..max to 0..1

reduction arises from around 15% of the decisions. If a manager cannot implement all SEESAW's recommendations and a 50% reduction is adequate, she might elect to use just these top 15% decisions. That is, SEESAW not only makes R recommendations, it also reports on the value of just applying just some subset of $r \subseteq R$.

In order to support incremental decision making, SEESAW uses a very large P value to control its reasoning. This P value is used to delay "lose-lose" decisions until the later iterations of the algorithm. Suppose our search has only constrained (say) $pcap = 5$ so far. Our next randomly chosen feature might be $data$, so SEESAW tries $\{Low, High\}$ values of $data = 2$ and $data = 5$. It is possible that both of these values will result in scores that are worse than the current score (using $pcap = 5$ alone). This means that $data$ is not a valuable feature to control, and picking either of the $\{Low, High\}$ values would be a "lose-lose" decision. Using $P = 0.95$, SEESAW will very likely (95% of the time) leave $data$ unconstrained and pick another, potentially better, feature to constrain on the next iteration. The algorithm will eventually come back to $data$, but if SEESAW delays making that lose-lose decision, it has a better chance of making good decisions early in the search.

6. The Models

This section describes the COCOMO effort Ef and time Ti predictor and the COQUALMO defect De predictor. SEESAW normalizes these scores 0..100 then seeks ways to minimize the following combination of the normalized scores: $\sqrt{Ef^2 + Ti^2 + De^2}$ (this expression is the Euclidean distance of normalized effort, time, defects to their minimum possible value).

Note that, in the following, the range of possible tunings within these models ins represented by Equations 4,5,7 and 8.

6.1. Effort Prediction with COCOMO

COCOMO predicts development effort Ef in total staff months where one month is 152 hours (and includes development and management hours). In COCOMO, the *scale factors* SF_i of Figure 3 affect effort exponentially on KSLOC (Thousands of Source Lines of Code) while *effort multipliers* EM_j affect effort linearly:

$$Ef = months = a * \left(KSLOC^{\left(b + 0.01 * \sum_{i=1}^{5} SF_i\right)} \right) * \left(\prod_{j=1}^{17} EM_j \right) \quad (2)$$

where KSLOC is estimated directly or computed from a function point analysis; SF_i and EM_j are the *scale factors* and *effort multipliers* of Figure 3; and a and b are the features discussed in Equation 1.

For effort multipliers, off-nominal ranges {vl=1, l=2, h=4, vh=5, xh=6} change the prediction by some ratio. The nominal range {n=3}, however, corresponds to an effort multiplier of 1, causing no change to the prediction. Hence, these ranges can be modeled as straight lines $y = mx + b$ 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:

$$\forall x \in \{1..6\}\ EM_i = m_\alpha (x - 3) + 1 \quad (3)$$

where m_α denotes the effect of effort multiplier a on *effort*.

The effort multipliers form into two sets:

1. The *positive effort EM* features, with slopes m_α^+, that are proportional to effort. These features are: cplx, data, docu, pvol, rely, ruse, stor, and time.
2. The *negative effort EM* features, with slopes m_α^-, are inversely proportional to effort. These features are acap, apex, ltex, pcap, pcon, plex, sced, site, and tool.

Based on prior work [5], we can describe the space of known tunings for COCOMO effort multipliers to be

$$\left(0.073 \le m_\alpha^+ \le 0.21\right) \wedge \left(-0.178 \le m_\alpha^- \le -0.078\right) \quad (4)$$

Similarly, using experience from 161 projects [5], we can say that the space of known tunings for the COCOMO scale factors (prec, flex, resl, team, pmat) are:

$$\forall x \in \{1..6\}\ SF_i = m_\beta (x - 6) \wedge (-1.56 \le m_\beta \le -1.014) \quad (5)$$

where m_β denotes the effect of scale factor i on *effort*.

One technical detail before continuing: we represent the space of possible tunings as minimum and maximum sloped lines that bound the space of known past tunings. Not all the COCOMO/COQUALMO relationships are linear; some are intricate piecewise linear functions. For those relationships, we extend our minimal and maximal lines such that the piecewise linear functions fall inside the extended bounds.

6.2. Time Prediction with COCOMO

COCOMO's *effort* prediction model relates to total staff time required for a project, COCOMO's *time* prediction model relates to project duration (in calendar months).

To predict time, COCOMO starts with the effort value Ef, removes the effect of schedule pressure, then raises the result to a fraction of the scale factors. Finally, an adjusted value $sced2$ is applied to the calculation:

$$Ti = c \cdot (Ef / sced)^{d + 0.2 \sum_i SF_i} \cdot sced2$$

The $\{c, d, sced2\}$ values come from tables of constants.

6.3. Defect Prediction with COQUALMO

COQUALMO has two core models, used three ways. The *defect introduction* model is similar to Equation 2: settings to Figure 3's effort multipliers and scale factors map to predictions about the number of defects that will be introduced in a given phase of development. Also, the *defect removal* model represents how various tasks (peer reviews, execution-based testing, and automated analysis) decrease the number of defects that flow through to the next stage of development.

These models are repeated for each of the three development phases of requirements, design, and coding. COQUALMO follows the same convention as COCOMO for the effort multipliers: nominal values ($n = 3$) do not change the predicted number of defects. Hence:

$$\forall x \in \{1..6\}\ Defects_i = m_\gamma(x - 3) + 1 \tag{6}$$

where m_γ denotes the effect of i on *defect introduction*. The effort multipliers and scale factors form two sets:

1. The *positive defect* features, with slopes m_γ^+, that are proportional to the estimated defects. These features are flex, data, ruse, cplx, time, stor, and pvol.
2. The *negative defect* features, with slopes m_γ^-, that are inversely proportional to the estimated defects. These features are acap, pcap, pcon, apex, plex, ltex, tool, site, sced, prec, resl, team, pmat, rely, and docu.

The space of tunings for defect introducing features are:

$$requirements \begin{cases} 0 \le m_\gamma^+ \le 0.112 \\ -0.183 \le m_\gamma^- \le -0.035 \end{cases}$$

$$design \begin{cases} 0 \le m_\gamma^+ \le 0.14 \\ -0.208 \le m_\gamma^- \le -0.048 \end{cases} \tag{7}$$

$$coding \begin{cases} 0 \le m_\gamma^+ \le 0.14 \\ -0.19 \le m_\gamma^- \le -0.053 \end{cases}$$

The space of tunings for defect removal features are:

$$\begin{array}{rl} \forall x \in \{1..6\} & SF_i = m_\delta(x - 1) \\ requirements: & 0.08 \le m_\delta \le 0.14 \\ design: & 0.1 \le m_\delta \le 0.156 \\ coding: & 0.11 \le m_\delta \le 0.176 \end{array} \tag{8}$$

where m_δ denotes the effect of i on *defect removal*.

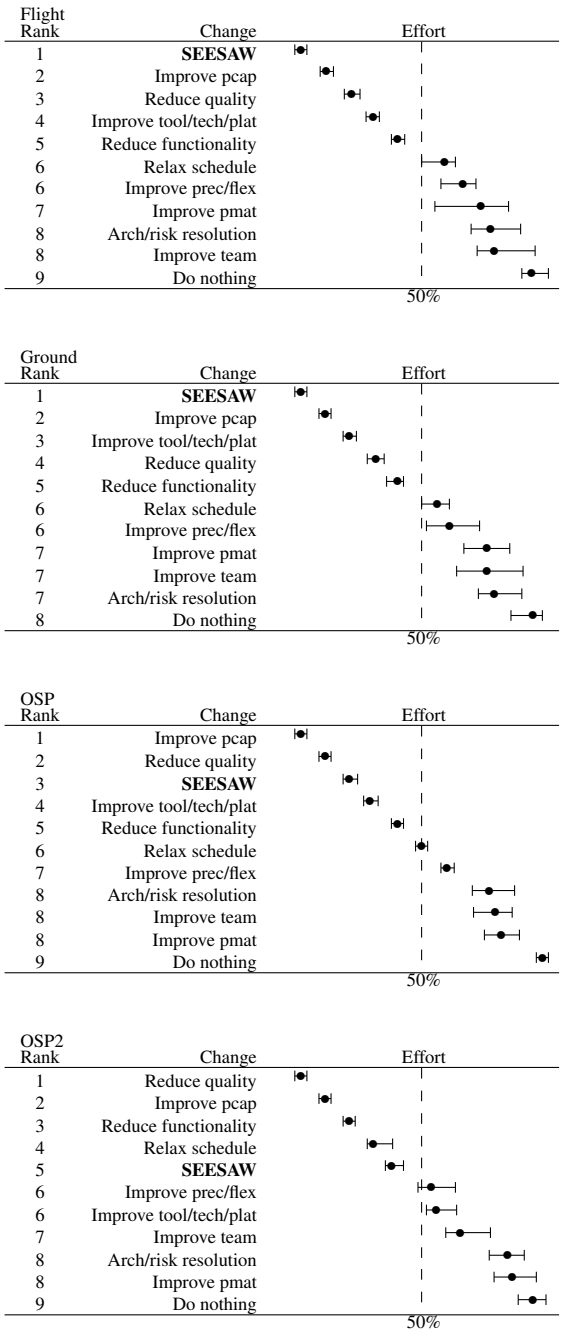

Figure 8. EFFORT: total staff months.

7. Results

Our experiments compared predictions after either (a) applying the drastic changes of Figure 1; or (b) constraining some internal options selected by SEESAW trying to minimize a linear combination of normalized defects, effort, and time; or (c) doing nothing at all and just running a Monte Carlo simulation across the ranges of Figure 4. Each experiment comprised 100 runs of Figure 6. Each time a set of project options were scored, m values were selected at random from Equations 4,5,7 and 8.

In Figures 8, 9, and 10, the rows are sorted by median scores. The "Do nothing" row comes from Monte Carlo simulations of the current ranges, without any changes. The

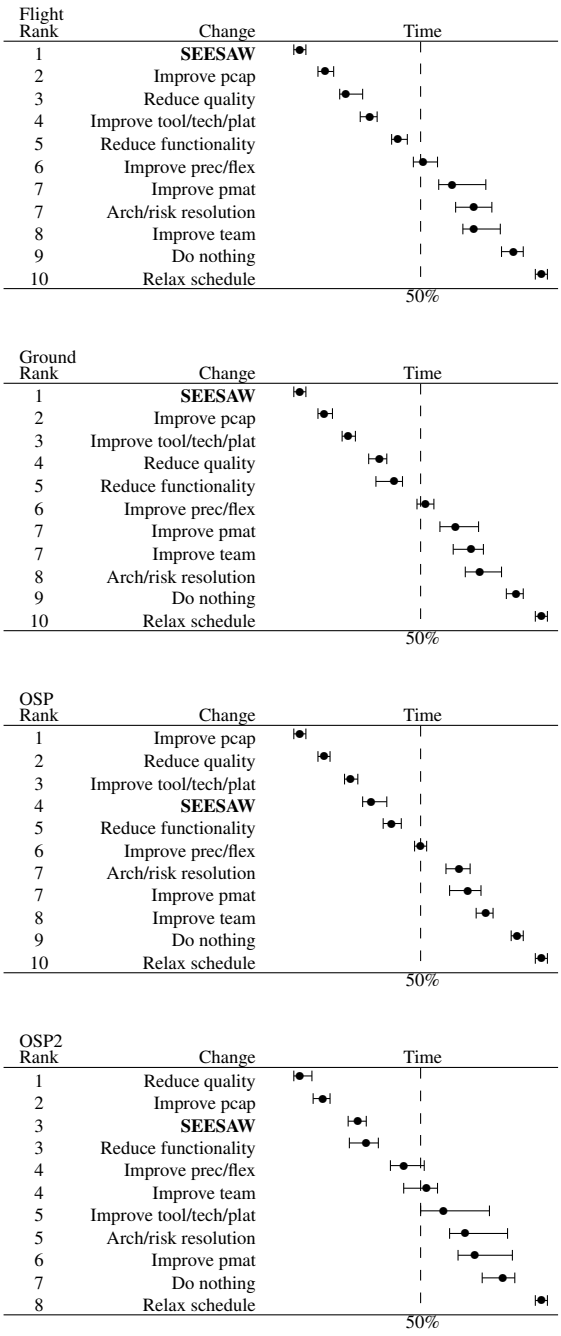

Figure 9. TIME: calendar months.

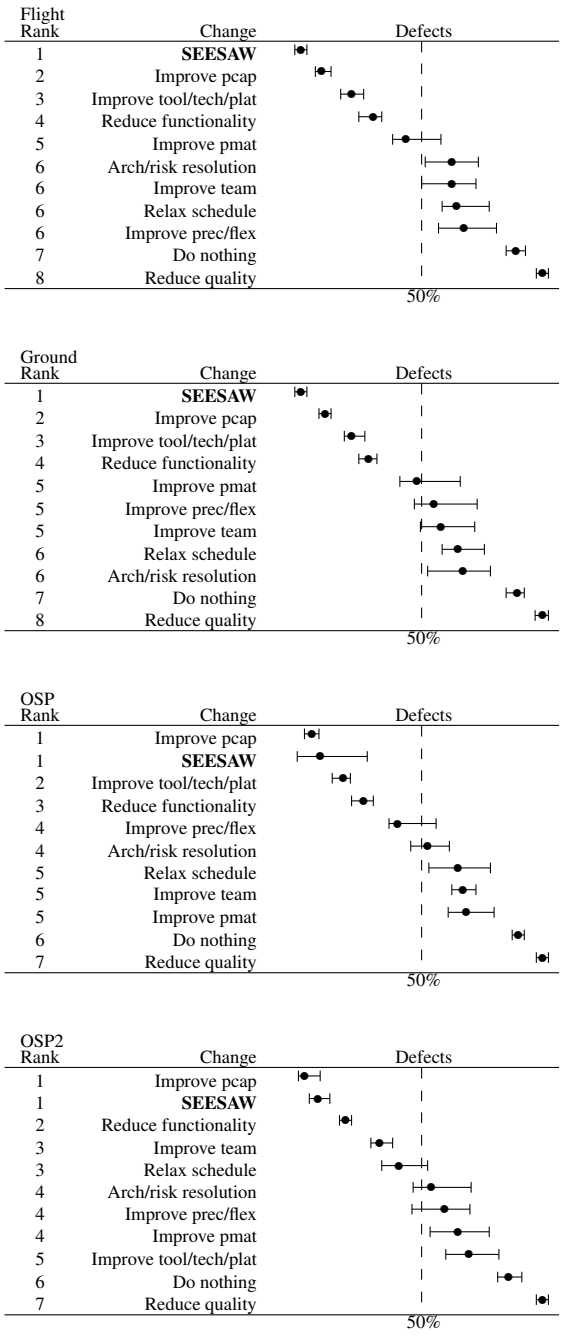

Figure 10. Defect / KLOC.

ranks shown in the left-hand column are computed from a Mann-Whitney test at 95% confidence test (this test was chosen since (a) due to the random nature of Monte Carlo simulations, the inputs to each run are not paired; and (b) ranked tests make no, possibly inappropriate, assumption about normality of the results). Two rows have the same rank if there is no statistical difference in their distributions.

Each row shows results from 100 `runs` (see Figure 6):

- All results are normalized to run 0..100, min..max.
- Each row shows the 25% to 75% quartile range of the normalized scores collected during the simulation.
- The median result is shown as a black dot.

All the performance scores (effort, time, defects) get *better*

Flight		Ground	
acap=5	100%	aexp=5	100%
aexp=5	100%	cplx=1	100%
cplx=3	100%	docu=1	100%
docu=1	100%	flex=5	100%
flex=5	100%	ltex=4	100%
ltex=4	100%	pcap=5	100%
pcap=5	100%	pcon=5	100%
pcon=5	100%	plex=4	100%
plex=4	100%	prec=5	100%
prec=5	100%	pvol=2	100%
pvol=2	100%	rely=4	100%
rely=5	100%	resl=5	100%
resl=5	100%	ruse=2	100%
ruse=2	100%	site=6	100%
site=6	100%	team=5	100%
team=5	100%	acap=5	95%
pmat=3	95%	time=3	95%
stor=3	90%	data=2	85%
etat=6	85%	stor=3	80%
data=2	80%	pmat=3	75%
time=3	75%	etat=1	60%
aa=6	55%	pr=6	60%
pr=6	50%	aa=1	55%
pr=1	45%	aa=6	45%
aa=1	40%	etat=6	35%
time=4	25%	pr=1	35%
data=3	15%	pmat=2	20%
etat=1	15%	stor=4	20%
stor=4	10%	data=3	15%
pmat=2	5%	acap=3	5%
		time=4	5%

Figure 11. SEESAW recommendations.

when the observed scores get *smaller*. Note that, usually, SEESAW obtains the better scores.

Figures 8, 9, and 10 show results from 12 experiments:

$$\{effort, time, defects\} * \{flight, ground, OSP, OSP2\}$$

In $\frac{12}{12}$ (i.e., all) experiments, the median score from SEESAW is better than at least half of the other changes. Better yet, in $\frac{6}{12}$ (i.e., half) experiments, SEESAW offers the best predictions, and in two more experiments (the defect results for OSP and OPS2), SEESAW ties the top rank. This result is the justification for our claim in the introduction that in a majority of experiments ($\frac{8}{12}$), SEESAW's search through the space of internal changes to current project options does as well or better than drastic change.

Another result of interest is that the range of SEESAW results is typically smaller than most other changes in Figures 8, 9, and 10 (observe how the 75% - 25% band in SEESAW's results are usually smaller than all the others). That is, of all the changes proposed here, the most stable were those generated by the stochastic search. This paradoxical result is simple to explain: SEESAW does not just *sample* a space of options; it also hunts for options that *reduces* the median score within those options. For models where the variance is proportional to the median, then reducing the median score also reduces the variance.

Another stable feature of SEESAW are the actual recommendations. An issue with stochastic sampling is that the output recommendations can vary dramatically from one run to another. Such variance can occur when a stochastic methods staggers between multiple solutions of near equal value. Figure 11 shows the recommendations made by 20 runs of SEESAW for the flight and ground projects. Note that the majority of the recommendations occur in all 20 runs and represent the most important actions that a project manager could make. The recommendations that occur at low frequencies are still valid, if used in conjunction with the other recommendations of that particular run.

The worst results from SEESAW are see in Figure 8 where the median effort found by SEESAW is near the middle of all changes. This result is hardly surprising. This result comes from OSP2 and this case study is the most constrained case explored by this paper (observer in Figure 4 that, for the most part, the features all have fixed values). SEESAW works by exploring subsets of the available options. When that option space is limited, the benefits of SEESAW are also limited. That is, we cannot find successful options unless we are first given enough options to explore. Previously [15] we have expressed this as "if you fix everything, there is nothing left to fix".

In other cases where drastic changes wins, those changes are often unacceptable. For example, in the effort and time results, SEESAW is beaten by "reduce quality" in $\frac{4}{12}$ experiments. The benefits of these effort and time reductions should be weighed against the implications on defects. Figure 10 shows that "reduce quality" results in the largest number of defects seen in any change. While there may exist projects that permit delivering code with large numbers of defects, we believe that usually this is unacceptable.

Another drastic change that defeats SEESAW is "improve pcap" (personnel capabilities). Recalling Figure 5, we see that improving personnel capability requires changing two analyst features, three personnel features, as well as addressing some issues relating to language and tool experience. Of all the drastic changes studied in this paper, this change is the most complex. It might also be the hardest to implement since it requires dramatic change to the current personnel working on the project (perhaps even firing old staff in order to hire new personnel). If an organization can handle such large-scale change, then in some cases (e.g., in $\frac{4}{12}$ experiments from Figures 8, 9, and 10) the "improve pcap" option might indeed outperform SEESAW. However, managers might want to avoid such complex organizational change, preferring instead to fine tune their project within its current structure using just internal changes.

8. Discussion

Our thesis is that there exist some process models (e.g. COCOMO/COQUALMO) that can still be used to rank proposed process changes, even without sufficient data to achieve precise local tunings. This is not to say that local

tuning should not be attempted. Indeed, our strong view is that the preferred option is to use precisely tuned models. However, in domains where such tunings are unavailable, we recommend SEESAW.

We show that for COCOMO/COQUALMO, prediction uncertainty can be mitigated by selecting the right project options *regardless* of uncertainty in the tunings. This is *not* to say that prediction uncertainty of *all* process models can be mitigated in the same manner. However, the observation that COCOMO/COQUALMO models can be controlled in this manner is one more reason to prefer these models.

Nor do we claim that stochastic stability *always* beats *all* drastic change for all projects. Indeed, we offer examples where extremely drastic change *defeats* the internal changes proposed by SEESAW. However, in those cases, those changes were so drastic (reduce functionality, or make major change the personnel) that many managers may prefer SEESAW's slightly-less-then-best recommendations.

9. Related Work

Much of the related work on uncertainty in software engineering uses a Bayesian analysis. For example, Pendharkar et.al. [18] demonstrate the utility of Bayes networks in effort estimation while Fenton and Neil explore Bayes nets and defect prediction [10] (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.

The process simulation community (e.g. Raffo [20]) studies models far more elaborate than COCOMO or COQUALMO. While such models offer more detailed insight into an organization, the effort required to tune them is non-trivial. For example, Raffo spent two years tuning one such model to one particular site [19].

Other related work is the search-based SE approach advocated by Harmon [11]. Search-Based Software Engineering (SBSE) uses optimization techniques from operations research and meta-heuristic search (e.g. simulated annealing and genetic algorithms) to hunt for near optimal solutions to complex and over-constrained software engineering problems. The SBSE approach can and has been applied to many problems in software engineering (e.g. requirements engineering [12]) but most often in the field of software testing [1]. Harmon's writing inspired us try simulated annealing to search the what-ifs in untuned COCOMO models [16]. However, we found that SEESAW ran much faster and produced results with far less variance than simulated annealing.

10. Future Work

The results of this paper assume that all the options found by SEESAW are *controllable*; i.e. a manager can implement all of SEESAW's recommendations. Future work should repeat this analysis assuming that only certain subsets of the internal options are controllable.

Also, this paper only explores the drastic changes of Figure 1 (these where defined in one of our prior publications [3]). There may be other drastic changes that defeat the recommendations of SEESAW. Future work should explore a wider range of drastic changes.

11. Conclusion

Drastic change is highly disruptive to a project. For example, your manager has returned from ICSE'09 excited about the next new thing in software engineering. "We need to drastically alter our projects," she says, "in order to support (say) automatic formal methods, agile process, etc.". This change may entail switching from Windows to LINUX (since there are more research prototypes for automated formal methods running on LINUX); stopping production while engineers retrain; or, worse, firing the engineers and making new hires.

An alternative to drastic change are combinations of internal changes within the current bounds of a project. Given precise local tunings to linear software process models (e.g. the COCOMO effort predictor and the COQUALMO defect predictors), then simple linear extrapolation may suffice for exploring different process options. However, when the tunings are not precise, or competing goals have to be achieved, simple linear extrapolation is insufficient.

This paper has explored the hard case of (a) ranking different process changes using (b) models with competing influences that (c) have not been precisely tuned using local data. In this case, the core task is to find *stability* within a very large space of possibilities. Given a project with some variance in its process options and a model with some variance in its tunings, then the model prediction uncertainty is some combination of the uncertainty with a project's options and uncertainty with the model's tunings. Local data can be used to reduce the uncertainty in the tunings. However, as shown in Equation 1, even after tuning it is possible tuning uncertainty remains.

Based on research dating back to 1981 [4], we assert that the space of *possible tunings* for COCOMO/COQUALMO is well defined (see Equations 4,5,7 and 8). We show here that at least for the these models, it is possible to explore a very large range of "what-ifs" (the space of all possible tunings and internal project changes) to find project option settings that improve model predictions, despite uncertainty in the tunings.

Specifically, in studies with 4 NASA case studies and 8 drastic changes, we found that, usually, SEESAW's stochastic sampling found internal options that worked as well or out-performed drastic change. Essential to the above is *stochastic stability*. Stochastic search enables the rapid sampling of a very wide range of options. More importantly, when studying the space of unknowns associated with a particular project (e.g. Figure 4), stochastic sampling can return multiple results after multiple runs. These results can be studied to check the *stability* of the solutions within the space of project options. Figures 8, 9, and 10 show that the SEESAW stochastic stability tool typically generated the smallest range in performance predictions. That is, of all the changes studied here, SEESAW's recommendations were typically the most stable.

Our results suggest that project managers may have more project improvement options than they currently realize. Hence, before performing drastic changes it can be worthwhile to check for stochastically stable effects amongst the internal changes, if only to understand better the available options within the current structure of a project.

Acknolwedgements

Thanks to Phillip Green, Topi Haapio and Andi Marcus and for their feedback on earlier drafts.

References

[1] J. Andrews, F. 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.

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

[3] B.Boehm and H.In. Conflict analysis and negotiation aids for cost-quality requirements. *Software Quality Professional*, 1(2):38–50, March 1999.

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

[5] 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.

[6] B. Boehm, C. Abts, and S. Chulani. Software development cost estimation approaches - a survey. *Annals of Software Engineering*, 10:177–205, 2000.

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

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

[9] J. Crawford and A. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *AAAI '94*, 1994.

[10] 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.

[11] M. Harman and J. Wegener. Getting results from search-based approaches to software engineering. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 728–729, Washington, DC, USA, 2004. IEEE Computer Society.

[12] O. Jalali, T. Menzies, and M. Feather. Optimizing requirements decisions with keys. In *Proceedings of the PROMISE 2008 Workshop (ICSE)*, 2008. Available from http://menzies.us/pdf/08keys.pdf.

[13] 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.

[14] T. Menzies, Z. Chen, J. Hihn, and K. Lum. Selecting best practices for effort estimation. *IEEE Transactions on Software Engineering*, November 2006. Available from http://menzies.us/pdf/06coseekmo.pdf.

[15] 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.

[16] T. Menzies, O. Elrawas, J. Hihn, M. Feathear, B. Boehm, and R. Madachy. The business case for automated software engineerng. In *ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 303–312, New York, NY, USA, 2007. ACM. Available from http://menzies.us/pdf/07casease-v0.pdf.

[17] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995. (reprinted 1997,2000).

[18] P. C. Pendharkar, G. H. Subramanian, and J. A. Rodger. A probabilistic model for predicting software development effort. *IEEE Trans. Softw. Eng.*, 31(7):615–624, 2005.

[19] D. 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.

[20] 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.