The Business Case for Automated Software Engineering

Tim Menzies
Oussama Elrawas
Lane Dept. CSEE
West Virginia University, USA
tim@menzies.us
oelrawas@mix.wvu.edu

Jairus Hihn
Martin S. Feather
Jet Propulsion Laboratory
California, USA
jairus.hihn@jpl.nasa.gov
mfeather@jpl.nasa.gov

Ray Madachy
Barry Boehm
Computer Science Dept.
Uni. Southern California, USA
madachy@usc.edu
boehm@sunset.usc.edu

ABSTRACT

Adoption of advanced automated SE (ASE) tools would be more favored if a business case could be made that these tools are more valuable than alternate methods. In theory, software prediction models can be used to make that case. In practice, this is complicated by the "local tuning" problem. Normally, predictors for software effort and defects and threat use local data to tune their predictions. Such local tuning data is often unavailable.

This paper shows that assessing the relative merits of different SE methods need *not* require precise local tunings. STAR1 is a simulated annealer plus a Bayesian post-processor that explores *the space of possible local tunings* within software prediction models. STAR1 ranks project decisions by their effects on effort and defects and threats. In experiments with NASA systems, STAR1 found one project where ASE were *essential* for minimizing effort/ defect/ threats; and another project were ASE tools were merely *optional*.

Categories and Subject Descriptors

I.6 [**Learning**]: Machine Learning; D.2.8 [**Software Engineering**]: Metrics—*product metrics, process metrics*

Keywords

COCOMO, COQUALMO, simulated annealing, Bayes

1. INTRODUCTION

Much current ASE research concerns *automatic analysis* of source code or better *execution-based testing tools*, These tools might, say, verify formal properties or search for the fewest regression tests that exercise most of the system. Some of these tools are ready for industrial use; e.g, SPIN [15] or JPF [14], just to name a few.

Recently, the authors were asked to make the business case for introducing some of these new ASE tools into large NASA projects. This case proved difficult to make, due to an *anti-automation bias* and the *local tuning problem*.

This research was conducted at West Virginia University, Univerity of Southern California, and NASA's Jet Propulsion Laboratory 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.

Submitted to ASE '07 Atlanta, Georgia, USA

The *anti-automation bias* was seen at an ICSE 2007 panel. Tim Lister (a co-author of Peopleware [8]) commented that "sociology beats technology in terms of successfully completing projects"- a notion endorsed by the other panelists. That is, software project managers should focus *less* on new ASE tools and *more* on managing the sociology aspects of their team (e.g. decrease staff turnover).

Figure 1 offers some support for this bias. This figure shows the known relative productivity effects of changing project features. According to this figure, the benefits of automatic tools (ranked number nine in the list) can be equaled or bettered via other means (e.g. any item 1 to 8 *or* any *pair* of items 10 to 22).

Note that this support for the anti-automation bias is based solely on the development effort; i.e. Figure 1 is blind to the impact of new ASE tools in reducing *defects* and any other *threats* to the success of the project. A complete business case should therefore study predictors for effort *and* detects *and* threats using, for example:

- The COQUALMO defect predictor [3, p254-268];
- The COCOMO effort predictor [3, p29-57];
- The THREAT predictor for effort & schedule overrun [3, 284-291].

Such a study is complicated by the *local tuning problem*. These predictors are most accurate *after* they have been tuned to local data (see §6). Unfortunately, the data required for local tuning is difficult to obtain, especially across multiple organizations [22]. This is

id	features	relative weight
1	Personnel/team capability	3.53
2	Product complexity	2.38
3	Time constraint	1.63
4	Required software reliability	1.54
5	Multi-site development	1.53
6	Doc. match to life cycle	1.52
7	Personnel continuity	1.51
8	Applications experience	1.51
9	Use of software tools	1.50
10	Platform volatility	1.49
11	Storage constraint	1.46
12	Process maturity	1.43
13	Language & tools experience	1.43
14	Required dev. schedule	1.43
15	Data base size	1.42
16	Platform experience	1.40
17	Arch. & risk resolution	1.39
18	Precedentedness	1.33
19	Developed for reuse	1.31
20	Team cohesion	1.29
21	Development mode	1.32
22	Development flexibility	1.26

Figure 1: Relative effects on development effort. Data from a regression analysis of 161 projects [2].

due to the business sensitivity associated with the data as well as differences in how the metrics are defined, collected and archived. In many cases the required data has not been archived at all. For example, after two years we were only able to add 7 records to our NASA wide software cost metrics repository.

The premise of this paper is that, in terms of ranking different methods, a *precise* local tuning is not required. Based on research dating back to 1981 [1], we assert that the space of *possible* tunings is well known. This space can be explored to find features that minimize effort, defects, and threats.

To test that premise, we built the STAR1 algorithm. STAR1 uses a simulated annealer to sample the space of possible local tunings within COCOMO, COQUALMO and THREAT. A standard SA offers constraints to all variables. STAR1, on the other hand, uses a Bayesian method to rank those constraints according to how well they reduce effort/ defects/ threats. Superfluous details can hence deleted and the algorithm can return a minimal set of project decisions that most reduce effort *and* defects *and* threats.

The general contributions of this paper, hence, are:

- A *novel search engine* for sampling, then pruning a space of options within a model.
- A *novel combination* of effort/defect/threat prediction models. Previously, these three models have been analyzed one at a time or in pairs [24, 26].
- A *new solution* to the local tuning problem: while waiting for local tuning data (which may never arrive), seek stable conclusions over the set of possible tunings.
- A *demonstration* that the relative merits of different project decisions can be assessed *without* specific local tuning data.

More specifically, using STAR1 we we found situations where new ASE tools were *optional* for one NASA systems but *essential* for another (measured in terms of reducing effort *and* defects *and* threats). Also, we did not find an *either/or* situation where we had to choose, as suggested by Lister, between sociology and technology. When ASE tools were useful, they were useful *in combination* with the sociological features.

The rest of this paper documents the COCOMO, COQUALMO, and THREAT models, along with the STAR1 algorithm. STAR1 is then applied to two NASA case studies. This will be followed by notes on external validity and related work.,

2. DEFINITIONS

Before we can assess the relative merits of new ASE tools versus other methods, we must first define "new ASE tools" and "other methods". Such a definition can be achieved via the ontology of COCOMO, COQUALMO, and THREAT, shown in Figure 2. This figure lists a variety of project *features* with the *range* {very low, low, nominal, high, very high, extremely high} or

$$\{vl = 1, l = 2, n = 3, h = 4, vh = 5, xh = 6\}$$

Lister's sociological features occur many times within Figure 2. For example, *team* refers to the sociology within a development team and its cohesiveness while *pcon* refers to the staff turnover rate within a project. Also listed in Figure 2 are factors like *acap, pcap* representing analyst and programmer capabilities (respectively).

As to technological features, new ASE tools appear as *execution-based testing tools* and *automated analysis*. Chulani [9] defines the top half of *automated analysis* as:

4 (high): intermediate-level module and inter-module code syntax and semantic analysis. Simple requirements/design view consistency checking.

5 (very high): More elaborate requirements/design view consistency checking. Basic distributed-processing and temporal analysis, model checking, symbolic execution.

6 (extremely high): Formalized[1] specification and verification. Advanced distributed processing and temporal analysis, model checking, symbolic execution.

The upper half of *execution-based testing and tools* is:

4 (high): Well-defined test sequence tailored to organization (acceptance / alpha / beta / flight / etc.) test. Basic test coverage tools, test support system.

5 (very high): More advanced test tools, test data preparation, basic test oracle support, distributed monitoring and analysis, assertion checking. Metrics-based test process management.

6 (extremely high): Highly advanced tools for test oracles, distributed monitoring and analysis, assertion checking Integration of automated analysis and test tools. Model-based test process management.

A review of recent proceedings of the IEEE ASE conferences suggests that a range of five or six in the above features includes the kind of "new ASE tools" explored at this venue. Hence, to compare "new ASE tools" to "other methods", we will try to reduce effort and defects and threats using just

$$\begin{array}{c} \textit{automated analysis} \in \{5, 6\} \vee \\ \textit{execution-based testing and tools} \in \{5, 6\} \end{array}$$

or "other methods" (i.e. other ranges of Figure 2).

3. THE MODELS

This section describes our predictors (for full details, see [25]). Each description includes notes on the space of known tunings within that model. In the sequel, STAR1 will explore that space.

3.1 Effort Prediction with COCOMO

COCOMO predicts development effort in calendar months where one month is 152 hours (and includes development and management hours). In COCOMO, the *scale factors* SF_i of Figure 2 effect effort exponentially (on KSLOC, i.e. Thousand Source Lines of Code) while *effort multipliers* EM_j effect effort linearly:

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

where KSLOC is estimated directly or computed from a function point analysis; SF_i and EM_j are the *scale factors*; *effort multipliers* of Figure 2; and a and b are domain-specific parameters. In our NASA data, these ranges

$$(3.72 \leq a \leq 9.18) \wedge (0.88 \leq b \leq 1.09) \quad (2)$$

With the effort multipliers, off-nominal ranges (i.e. {vl=1, l=2, h=4, vh=5, xh=6}) change the prediction by some amount. In the special case of the nominal range (i.e. {n=3}), that amount is one; i.e. the nominal range make no change to the prediction. Hence, these ranges can 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:

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

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

The effort multipliers form into two sets:

[1]Consistency-checkable pre- conditions and post-conditions, but not necessarily mathematical theorems.

	Definition	Low-end = {1,2}	Medium ={3,4}	High-end= {5,6}
Defect removal features				
execution-based testing	all procedures and tools used for testing	none	basic testing at unit/ integration/ systems level; basic test data management	advanced test oralces, assertion checking, model-based testing
automated analysis	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	all peer group review activities	none	well-defined sequence of preperation, informal assignment of reviewer roles, minimal follow-up	formal roles plues extensive review checklists/ root cause analysis, contnual 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 risk 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 15%	55%	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 2: Features in COCOMO, COQUALMO, and THREAT.

1. The *positive effort EM* features, with slopes m_a^+, 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_a^-, are inversely proportional to effort. These features are acap, apex, ltex, pcap, pcon, plex, sced , site, tool.

Based on prior work [2], we can describe the space of known tunings for COCOMO effort multipliers to be

$$\big(0.073 \leq m_a^+ \leq 0.21\big) \wedge \big(-0.178 \leq m_a^- \leq -0.078\big) \quad (4)$$

Similarly, using experience from 161 projects [2], we can say that the space of known tunings for the COCOMO scale factors (prec, flex, resl,team, pmat) are:

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

where m_b denotes the effect of scale factor b on *effort*.

Note that the above ranges for the slopes were obtained by finding the average slope for each COCOMO attribute for both effort multipliers and scale factors over the range of values of that attribute.

3.2 Defect Prediction with COQUALMO

COQUALMO has two core models, used three ways:

- The *defect introduction* model is similar to Equation 1; i.e. settings to Figure 2's effort multipliers and scale factors map to predictions about number of defects.
- The *defect removal* model represents how various tasks (peer review, execution-based testing, and automatic analysis) decrease the introduction of defects.
- The above two models are repeated three times for defects introduction & removal for requirements, design, or coding.

COQUALMO follows the same convention as COCOMO for the effort multipliers; i.e. nominal values ($n = 3$) add nothing to the predicted number of defects. As above, COQUALMO is:

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

where m_c denotes the effect of c on *defect introduction*.

The effort multipliers and scale factors form two sets:

1. The *positive defect* features, with slopes m_c^+, that are proportional to the estimated introduced defects. These features are flex, data, ruse, cplx, time, stor, and pvol.

	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

Figure 3: An example risk table

2. The *negative defect* features, with slopes m_c^-, that are inversely proportional to the estimated introduced 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:

$$
\begin{aligned}
requirements &\left\{ \begin{array}{c} 0 \le m_c^+ \le 0.112 \\ -0.183 \le m_c^- \le -0.035 \end{array} \right. \\
design &\left\{ \begin{array}{c} 0 \le m_c^+ \le 0.14 \\ -0.208 \le m_c^- \le -0.048 \end{array} \right. \\
coding &\left\{ \begin{array}{c} 0 \le m_c^+ \le 0.140 \\ -0.19 \le m_c^- \le -0.053 \end{array} \right.
\end{aligned}
\tag{7}
$$

The space of tunings for defect removal features are:

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

where m_d denotes the effect of d on *defect removal*.

3.3 THREAT: Predicting Effort and Schedule Overrun

The THREAT model returns a heuristic estimate of the threat of a schedule over run in the project. This estimation model is dependent upon the COCOMO effort multipliers

Internally, THREAT contains dozens of tables of the form of Figure 3. Each such table adds some "threat" value to the overall project risk when multiplied by the effort multiplier values of the corresponding COCOMO attributes. There are six major categories: schedule, product, personnel, process, platform and reuse. After the threat for each category is calculated, the sum is normalized to produce the final threat rating.

Figure 3 can be represented as an exponentially decaying function that peaks in one corner of the risk table at a value of two. All the tables peak at either a value of two or four. Since this model is heuristic in nature, the exact height of the peak is not certain. When we perform Monte Carlo simulations over THREAT, we vary the height of the peak by a random factor $0.5 \le x \le 1$ if the peak is four, and $0.5 \le x \le 1.5$ if the peak is two.

4. SEARCH ALGORITHMS

Having defined the internal space of our models, we now search it. A simulated annealer is used to sample that space and a back select algorithm is used to rank feature ranges according to how well they reduce effort/ defects/ threats.

4.1 Simulated Annealing

Monte Carlo algorithms randomly sample the space of possible controllable model states. A *Metropolis* Monte Carlo algorithm [27] creates new states by small mutations to some *current* state. If a new state is "better" (as assessed via an *energy function*), it becomes the new *current* state used for future mutations. Otherwise, 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. The algorithm is silent on the mutation mechanism. For our experiments, we freeze $\frac{2}{3}$ of the features and randomly select ranges for the rest.

```
function sa(kmax)
  s := s0; e := E(s)           // Initial state, energy.
  sb := s; eb := e             // Initial "best" solution
  k := 0                       // Energy evaluation count.
  while k < kmax and e > emax  // Loop
    sn := neighbour(s)         //   Pick some neighbour.
    en := E(sn)                //   Compute its energy.
    if en < eb then            //   Is this a new best?
      sb := sn; eb := en       //     Yes, save it.
    if random() < P(e, en, temp(k/kmax))
     then s := sn; e := en     //     Maybe jump
    k := k + 1                 //   One more evaluation done
  return sb                    // Return best
```

Figure 4: SA pseudo-code: a new solution sn (with new energy en) replaces the current solution if (a) it has a lower energy or (b) the acceptance predicate P endorses it. Only in the case of (a) should the new solution replaces the current best solution.

In 1983, Kirkpatrick et.el. [20] proposed a modification that was inspired by a procedure used to make the strongest possible glass. Initially, glass is heated to allow atoms to move freely. The temperature is then slowly lowered such that the atoms can find the stablest configuration with lowest energy. A *simulated annealing* (SA) algorithm adds a "temperature" variable to the Boltzmann accept criteria such that, at high temperatures, it is more likely that the algorithm will 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.

With SA, better solutions have lower "energy" so the fitness function E is defined such that $E \ge 0$ and *lower* values are $better$. For our purposes, we model effort (Ef) as the distance to the origin of the 3-D space with defects (De) and threats (Th):

$$
E = \left(\sqrt{\overline{Ef}^2 + \overline{De}^2 + \overline{Th}^2} \right) / \sqrt{3} \tag{9}
$$

Here, $\overline{x}$ is a normalized value $0 \le \frac{x - min(x)}{max(x) - min(x)} \le 1$. Hence, our energy ranges $0 \le E \le 1$ and *lower* energies are *better*.

The acceptance criteria P for a new state is defined using the current temperature T, the energy of the current solution (e), and the energy of the new mutation (en):

$$
P(e, en, T) = e^{(e-en)/T} \tag{10}
$$

T is defined to $decrease$ as the simulator loops through $k = 1 \ldots kmax$ iterations. We use $T = e^{-100 * k/kmax}$.

Two advantages of SA are their *implementation simplicity* and their *ability to handle non-linear models*:

- *Implementation simplicity:* Figure 4 illustrates the simplicity of the algorithm. Memory is only required for one current solution (s), one new solution (sn) and one best solution (sb) that stores the best solution seen at any time in the simulation.
- *Non-linear models:* Previously [7,11], we have applied SA to non-linear JPL requirements models where minimizing the cost of project mitigations can decrease the number of requirements achieved by that project. Hence, decreasing *both*

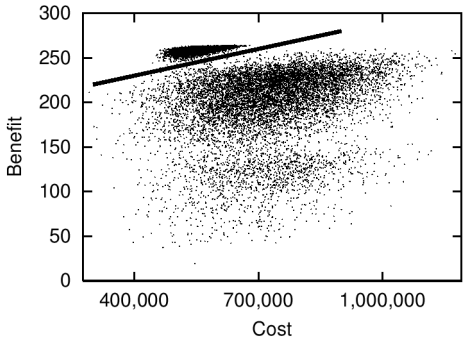


Figure 5: Processing a JPL requirements model.

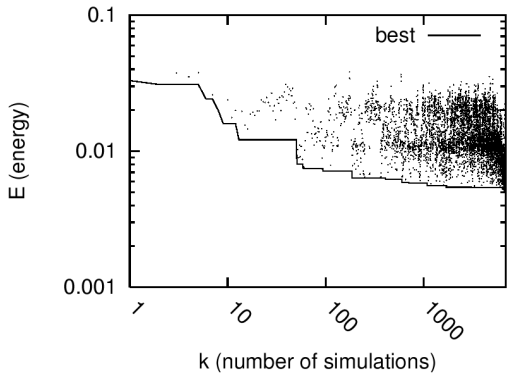


Figure 6: Dots & lines are SA output from current & best solution (respectively) after k simulations.

the cost and achieved requirements is a non-linear problem that must trade between minimizing cost and increasing requirements coverage. The top-left line of Figure 5 divides the behavior of the JPL requirements models before and after simulated annealing. As shown below the line, initial Monte Carlo sampling of the possible mitigations lead to a large range of costs and benefits. Simulated annealing found a set of mitigations that lead to the small cloud of solutions above the line. Compared to the initial samples, these new solutions had decreased cost, increased benefit (number of requirements covered), and decreased variance (shrank the space of solutions).

Two disadvantages of SA algorithms are their *incompleteness* and the *complexity* of their solutions:

- *Incompleteness:* In our domain, we have some evidence that the incomplete nature of the heuristic SA search is not a major problem. Figure 6 shows a sample run of our SA tool running on our prediction models for $K = 10,000$ simulations. As k increases for $1 \ldots 10,000$, it becomes less and less likely that a better $best$ has been missed. Hence, we run our simulations for ten times the period it takes for $best$ to stabilize (at $k \approx 1000$).
- *Solution complexity:* Simulated annealers offer constraints to all controllable features. Often this is an over-constrained solution since, in many domains, a repeated empirical result is a *feature subset selection* effect; i.e. models that constrain M variables perform just as well, or better, than models that constrain N variables ($M \ll N$) [12, 21]. For example, Kohavi [21] studied some machine learners to find that using just 19% of the available features *increased* prediction accuracy by just 2.14% (on average). For another example, when feature subset selection was applied to the JPL requirements model of Figure 5, we found that up to $\frac{2}{3}$-rds of the features can be left unconstrained, without effecting the conclusions [11].

In terms of the goal of this paper, exploring feature subset selection is very important. Before we can *remove* non-essential features, we must first *rank* them according to their effectiveness. As we shall see, this ranking will be insightful to the task of assessing the relative value of new ASE tools and other methods for reducing effort/ defects/ threats.

4.2 Support-Based Bayesian Ranking

STAR1 ranks the features ranges seen in K runs of a simulated annealer by dividing the K runs into:

- *Best:* those associated with the BEST% solutions (i.e. those with the BEST% *least* energy);
- And the *rest* (i.e. the other 100-BEST% of solutions).

It then computes the probability that a range is found in $best$. using Bayes' Theorem. Informally, the theorem says $next = old * new$ i.e. what we'll believe *next* comes from how *new* evidence effects *old* beliefs. More formally:

$$P(H|E) = P(E|H)P(H) \ / \ P(E) \tag{11}$$

i.e. using evidence E and a prior probability $P(H)$ for hypothesis $H \in \{best, rest\}$. The theorem calculate a posteriori probability $P(H|E)$. Simple Bayes classifiers are often called "näive" since they assume independence of each feature. While this assumption simplifies the implementation (frequency counts are required only for each feature), it is possible that correlated events are missed by this "näive" approach. Domingos and Pazzani show theoretically that the independence assumption is a problem in a vanishingly small percent of cases [10]. This explains the repeated empirical result that, on average, seemingly näive Bayes classifiers perform as well as other seemingly more sophisticated schemes (e.g. see Table 1 in [10]).

When applying the theorem, *likelihoods* are computed from observed frequencies, then normalize to create probabilities (this normalization cancels out $P(E)$ in Equation 11, so it need not be computed). For example, after $K = 10,000$ runs divided into 1,000 lowest 10% *best* solutions and 9,000 *rest*, the range $rely = vh$ might appears 10 times in the $best$ solutions, but only 5 times in the $rest$. Hence:

$$\begin{aligned}
E &= (reply = vh) \\
P(best) &= 1000/10000 = 0.1 \\
P(rest) &= 9000/10000 = 0.9 \\
freq(E|best) &= 10/1000 = 0.01 \\
freq(E|rest) &= 5/9000 = 0.00056 \\
like(best|E) &= freq(E|best) \cdot P(best) = 0.001 \\
like(rest|E) &= freq(E|rest) \cdot P(rest) = 0.000504 \\
P(best|E) &= \frac{like(best|E)}{like(best|E) + like(rest|E)} = 0.66
\end{aligned} \tag{12}$$

Previously [6] we have found that Equation 12 is a poor ranking heuristic since it is distracted by low frequency evidence. For example, note how the probability of E belonging to the best class is moderately high even though its support is very low; i.e. $P(best|E) = 0.66$ but $freq(E|best) = 0.01$.

To avoid such unreliable low frequency evidence, we augment Equation 12 with a support term. Support should $increase$ as the frequency of a range $increases$, i.e. $like(x|best)$ is a valid support measure. STAR1 hence ranks ranges via

$$P(best|E) * support(best|E) = \frac{like(x|best)^2}{like(x|best) + like(x|rest)} \quad (13)$$

4.3 The STAR1 Algorithm

To apply Equation 13, STAR1 runs in six phases. In terms of standard machine learning theory, step 1 generates a training set; steps 2,3,4 do some generalization; and step 5 tests the learned theory on data not seen during training.

1. *SAMPLE:* To sample the ranges from the models, STAR1 runs the simulated annealer $K_1 = 10,000$ times.

2. *DISCRETIZE:* The data seen in the K_1 samples is then discretized into $D = 10$ bins. Discretization converts a continuous range into a histogram with n break points $b_1 \ldots b_n$ where $(\forall i < j : b_i \leq b_j)$. After discretization, many observations can fall into the same range between b_i and b_{i+1} at frequency counts c_i. This study used equal width discretization; i.e.

$$\forall i, j : (b_i - b_{i-1}) = (b_j - b_{j-1})$$

3. *CLASSIFY:* The ranges are then classified into those seen in BEST% *best* or *rest*.

4. *RANK:* The ranges are then ranked in increasing order by their probability*support (Equation 13) of appearing in the $best$ outcomes;

5. *PRUNE:* STAR1 then runs K_2 experiments with the models where the top ranked ranges $1..X$ ranges are pre-set and the remaining ranges can be selected at random. In terms of the business case for automated software engineering tools, this step is crucial since it would rank new ASE tools alongside other methods.

6. *REPORT:* STAR1 returns the $1..X$ ranges that produce the least effort, defects, and threats.

To run our experiments, we had to apply our engineering judgment to set the parameters:

$$K_1 = 10,000, K_2 = 10,000, D = 10, BEST = 10$$

Initially, we planned experiments that varied these parameters, as well as trying other discretization policies. However, our initial results were so promising (see below) that we are not currently motivated to explore other parameter settings (such an exploration is planned for future work).

5. EXPERIMENTS

STAR1 was tested on the two case studies of Figure 7:

- OSP: The Orbital Space Plane GNC prototype (a 1990s NASA system). OSP was an early prototype for OSP2.
- OSP2: the guidance and navigation control system of a current NASA launch vehicle, under development.

In Figure 7, *values* are *fixed* while *ranges* represent a space of options. In the sequel, this observation will be important: OSP2 constrains most of its features to fixed values while OSP allows more variation in feature ranges.

	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

Figure 7: Two case studies.

SA was run K_1 times and ranges not mentioned in Figure 7 selected at random from Equations 2,4,5,7,8 and §3.3. The selected *controllable* ranges were ranked with Equation 13:

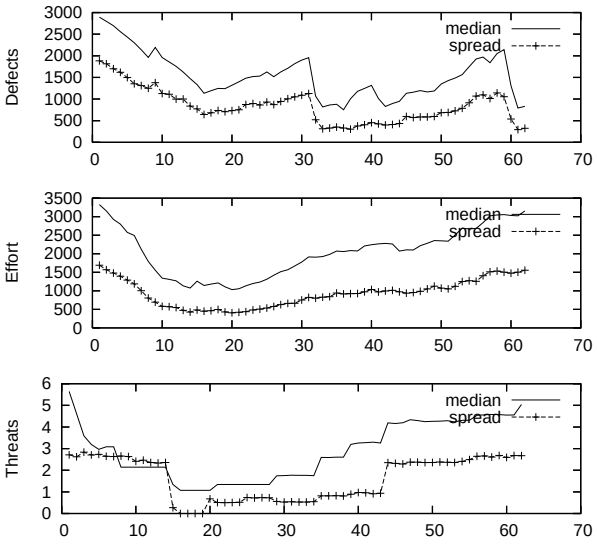
- By *controllable* features, we mean the kinds of changes a project manager could make to her project; i.e. Figure 2.
- *Uncontrollable* features are features like the correlation between, say, analyst capability and development effort; i.e. the internal model parameters of Equations 2,4,5,7,8 and §3.3.

The list at the bottom of Figure 8 shows how Equation 13 ranked OSP project decisions. These ranks correspond to the X-axis of the plots at the top of that figure. "Median" plots the 50% percentile of the defect/ effort/ threat values seen after imposing ranges $1..X$ (and selecting all other ranges at random). "Spread" shows the 75%-50% percentile range. These plots are U-shaped:

- On the left-hand-side of each plot, poor results were seen after applying too few constraints. That is, models perform poorly if we do not control them enough.
- On the right-hand-side of each plot, poor results were also seen. In terms of Equation 13, these are the ranges with low support and low probability of belonging to $best$ class. Hence, it is not surprising that applying these superfluous constraints was counter-productive.

The ranges from new ASE tools (see the "▲" symbol) are ranked at X=31, 37, 40, 41. These rankings can assessed using the minimum points in the median plots:

- At $X \approx 15$ effort and threat are at their global minimums while defects is at a local minimum.
- The global minimum for defects can be see at $X = 33$. At this point, defects are 30% less than at $X = 15$. However,

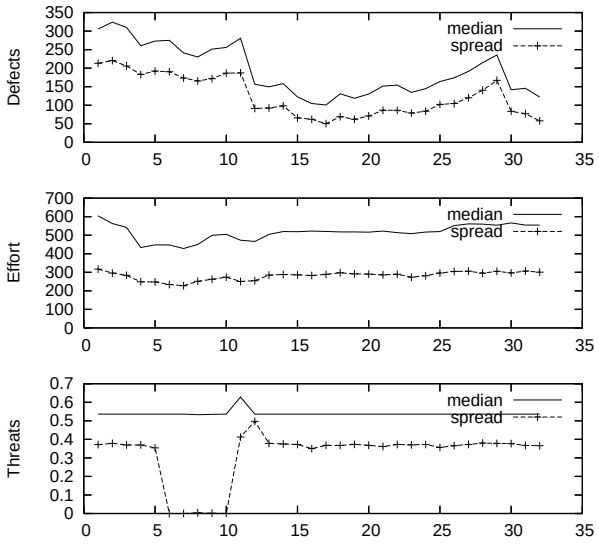


X	feature = range	
1	acap = 3	
2	cplx = 5	
3	tool = 3	
4	apex = 3	
5	pcon = 3	
6	ruse = 2	
7	time = 3	
8	docu = 2	
9	ltex = 4	
10	prec = 2	
11	team = 3	
12	stor = 3	
13	resl = 3	
14	sced = 3	
15	pmat = 4	
16	team = 2	
17	prec = 1	
18	flex = 5	
19	sced = 2	
20	pmat = 3	
21	pcon = 2	
22	stor = 4	
23	flex = 4	
24	apex = 2	
25	docu = 3	
26	ltex = 3	
27	resl = 2	
28	tool = 2	
29	time = 4	
30	ruse = 3	
31	execution testing and tools = 6	▲▲
32	peer reviews = 5	
33	resl = 1	
34	cplx = 6	
35	peer reviews = 6	
36	automated analysis = 2	
37	execution testing and tools = 5	▲▲
38	acap = 2	
39	pmat = 2	
40	automated analysis = 5	▲▲
41	automated analysis = 6	▲▲
42-62	not shown (for space reasons)	

Figure 8: OSP results

this lower defect level is achieved via doubling effort from 1000 to 2000. months.

Figure 8 shows that OSP needs some tool support. The third highest ranked range (X=3) is for $tool = 3$; i.e. medium range tool use, This corresponds to some support for edit, code, and debug moderately integrated with processes and methods and reuse.



X	feature = range	
1	docu = 3	
2	pmat = 5	
3	ltex = 5	
4	pmat = 4	
5	sced = 3	
6	prec = 5	
7	docu = 4	
8	ltex = 4	
9	prec = 4	
10	sced = 2	
11	automated analysis = 6	▲▲
12	sced = 4	
13	prec = 3	
14	execution testing and tools = 5	▲▲
15	execution testing and tools = 6	▲▲
16	peer reviews = 4	
17	automated analysis = 4	
18	peer reviews = 5	
19	execution testing and tools = 4	
20	peer reviews = 2	
21	automated analysis = 5	▲▲
22	peer reviews = 6	
23	peer reviews = 3	
24	execution testing and tools = 2	
25	ltex = 3	
26	automated analysis = 3	
27	automated analysis = 2	
28	execution testing and tools = 1	
29	execution testing and tools = 3	
30	peer reviews = 1	
31	automated analysis = 1	
32	ltex = 2	

Figure 9: OSP2 results.

However, Figure 8 makes only a weak business case for more extensive tool support. *Automated analysis* is ranked at positions 40,41 and is not required to reach the minimums of the Figure 8 plots. On the other hand, if management has the budget to support the X=33 effort, then using extremely high range *execution-based testing and tools* is strongly recommended.

Clearly, if the project is cost-adverse, then methods other than new ASE tools are recommended. However, the OSP business users should be aware of the consequences of this policy. Applying only decisions $X = 1..15$ will produce a product costing half of the previous recommendation, but with 30% more defects.

While new ASE tools are *optional* for OSP, they are *essential* for OSP2. Figure 9 shows the OSP2 results. Minimum defects were

achieved at X=17 via a *combination* of:

- *technical features*: project decisions from the range $1 \leq X \leq 17$ use three of our four new ASE tools (see X=11,14,15).
- and *sociological features* such as $docu$, use of documentation; $pmat$, process maturity; $ltex$, language & tool experience; and controlling $sced$ schedule pressure and the $prec$ precedentedness of the project.

Hence, the argument quoted in the introduction that "sociology beats technology" should be corrected to "sometimes, sociology can use more technology". For example, in Figure 9, only sociological features are applied for $1 \leq X \leq 10$. Observe how, at X=11, the largest drop in defects was achieved *after* applying extremely high range automated analysis.

Two other noteworthy features of Figure 9 are:

- The U-shapes seen in all the OSP results of Figure 8 only appears in OSP2's defect plot. The median effort and threat curves for OSP2 are essentially flat. As seen in Figure 7, OSP2 constrains more features than OSP. Hence, at least for OSP2 effort and threats, STAR1 could not change those median values.
- On the other hand, the flatness of the effort and threats plots means that they are stable across a wide range of project settings and internal variations to the prediction models. For example, according to this study, the effort for OSP2 has 50% to 75% range of 600 to 900 person months (median plus spread). Such stability is very useful for project planning.

6. EXTERNAL VALIDITY

One benefits of STAR1's analysis is a reduction in *sampling bias*. Conclusions reached from prediction models tuned to local data are biased by that tuning. The conclusions reached here, on the other hand, are stable over a large space of possible tuning biases.

Another issue is *model bias*. The above conclusions are drawn from the internal space of some software prediction models. Clearly, if those predictions models are wrong, then these conclusions are also wrong, For example, the reader might believe that this analysis has overlooked the impact of features *not* mentioned in Figure 2.

One mitigation for this risk is to use the best available models. The models used here have been extensively validated:

- Chulani et.al. [4] reports a one study with a Bayesian tuning algorithm using 161 projects. After tuning, a cross-validation study showed that COCOMO produced effort estimates that are within 30% of the actuals, 69% of the time.
- Studies with the COCOMO-81 project database have shown that the THREAT index correlates with well with the $\frac{months}{KDSI}$ (KDSI= thousand of delivered source lines of code). This result is consistent with the the base premise of THREAT; i.e. bad management can delay software delivery.
- COQUALMO was developed using extensive feedback from the COCOMO affiliates group. This group comprises dozens of companies, that have donated 161 data sets, that meet each year to discuss improvements to the current set of models[2].

Another threat to external validity are the biases of the authors. Given the intended publication venue, and the authors' history of publishing on automatic SE tools, perhaps this study is inherently biased in favor of new ASE tools.

To enable other researchers to question our biases, we take care to make our case studies fully reproducible. Reproducibility is an important methodological principle in other disciplines since it allows a community to confirm, refute, or even improve prior results. In our view, in the field of software engineering, there are all too few examples of reproduced, and extended, results. Accordingly, an appendix to this paper details how to install and run the software required to repeat the OSP&OSP2 case studies. Such reproducibility allows other researchers to check if our advocacy of automated tools biases our conclusions.

Yet another threat to external validity is our *evaluation bias*. Tacit in Equation 9 is an assumption that reductions in threat, effort, and defects are of *equal* value. This may not be the case, especially for safety critical software where reducing defects has top priority, regardless of the cost.

Having documented this problem, we now ignore it. This paper has shown that there exists at least one evaluation bias under which ASE tools are essential to at least one project. For this paper, that will suffice.

The final threat to external validity is STAR1's *search bias*. There are 2^N possible combinations of N ranges but STAR1's PRUNE operator only explores N of them. Hence, in theory, the rankings of Figure 8 and Figure 9 may not accurately reflect the true relative effectiveness of the ranges.

There is some theoretical and empirical evidence that this search bias is not a major issue. Clark has conducted a theoretical study concluding that Equation 13 offers the wrong rankings in a very small percent of cases [6]. To check his theory, Clark ran numerous experiments comparing (a) the estimated "best" combinations of ranges found by Equation 13 and the (b) actual best combinations found by generating combinations, then trying each one against the training data. In all those experiments, Clark only found one minor case where $ranking(a) \neq ranking(b)$.

Clark's results explains the U-shape seen in the above plots. If the rankings of Equation 13 were accurate then, initially, performance should improve as an increasing number of effective ranges are applied. Eventually we run out of the effective ranges and start applying ranges with low probability and support of belonging to the $best$ class. After that point, performance should degrade. In the middle of these two extremes would be a low energy valley where the most effective constraints have been applied. Such U-shape performance plots were seen in Figure 8 and Figure 9.

Clark's analysis is too long to present here. However, a small simulation illustrates his style of argument. STAR1's PRUNE operator ranks N ranges in the order $1, 2, 3..N$. Consider a *greedy search* that returns the first $1, 2, 3...X \leq N$ of these ranges. Such a search grows X till some superset $1..Y$ performs worse the subset $1..X$ ($X < Y$). For this greedy search to be optimal, then longer combinations $1..Y$ can't be more effective than shorter combinations $1..X$.

This condition can be checked via simulation. Consider the case of $N = 100$ ranges classified into 1000 $best$ cases and 9000 $rest$ cases; i.e. $P(best) = 10\%$, and $P(rest) = 90\%$. A simulator can randomly generates N=100 pairs $\{a, b\}$ for $F(a|best)$ and $F(a|rest)$ (respectively) where

$$1 \leq a \leq 1000 \wedge 1 \leq b \leq 9000$$

After Equation 13 ranks the pairs, the simulator picks two combinations $1..X$ and $1..Y$ where $1 \leq X < Y \leq 100$. According to Equation 12, 98% of time, the shorter combination has a higher probability than the longer one of being *best*. That is, STAR1's linear search over Equation 13's rankings 1 to N will not miss interesting combinations.

[2] http://sunset.usc.edu/events/2006/CIIForum/

7. RELATED WORK

This report is somewhat at odds with a standard IEEE ASE paper. For example, there many reports at this conference of innovative tools that find defects missed by other methods. Such reports offer strong support for the utility of new ASE tools. However, note that they assess the new ASE tools *soley* in terms of defect detection. In the paper, on the other hand, we have tried to assess new ASE tools in the terms of multiple success criteria (defect removal, effort, threats).

This paper was motivated by a lack of results regarding the relative merits of new ASE tools versus different methods. To be fair, apart from Figure 1, the general software engineering literature is weak on the relative merits of *any* set of methods. We recently reviewed the (approx) 100 V&V methods offered by the SE literature[3]. While some evidence was offered for the value of general techniques[4] there was very little precise evidence on the merits of a particular tool. Further, there was almost no comparisons of the relative effectiveness of pairs of methods applied to the same task.

Lacking results from the literature, we turned to model-based methods. This work is an example of the Harman's *search-based software engineering* (SBSE) paradigm [5, 13] where SE activities are recast as optimization problems. STAR1 searches COCOMO-family software prediction models. Numerous alternate models have been developed by the software prediction and modeling community[5]. Some of these are specialized models are built to serve the needs of a particular software development organization. The COCOMO family of models, on the other hand, were built to generalize across the community of the COCOMO affiliates.

Historically, this work was inspired by Josephson and Chandrasekaran's work on exploration of large design spaces [18]. In 1999, Josephson reviewed our earliest prototype, which was taking too long to terminate. His proposed solution, (try some stochastic sampling) eventually grew into STAR1.

We have recently become aware that STAR1's Bayesian ranking methods is analogous to Ruinstein's cross-entropy method [28]. Cross-entropy is applied during simulated annealing to speed up convergence to best solutions (by avoiding rare events). In future work, we will explore more the connection of our Bayesian ranking to cross-entropy.

Prior related work by this team includes:

- The TAR1 minimal contrast set learner, reported at ASE 2000, explored a Monte Carlo simulations of the COCOMO and THREAT models [26].
- TAR2 [23], an optimized version of TAR1, was applied at ASE 2002 [24] and RE'02 [11] to different software prediction models.

This paper reports numerous improvements over this prior work:

- Previously, we worked on a single prediction model or pairs of predictive models. This paper is the first to study effort *and* defects *and* threats in one combined analysis.
- This prior work did not adjust the parameters internal to the prediction model. This new work, on the other hand, makes extensive adjustments to those parameters.
- Previously, our learners commented on just a small subset of the ranges. This new work ranks all the ranges, allow managers to perform their own detailed analysis of the relative effects of different project decisions.
- This new learner runs much faster than the prior work. STAR1 generated the OSP and OSP2 results in less than a minute. A similar range ranking, by TAR1, would require an overnight run [26]. TAR2 (and a later version, TAR3) run much faster but even those optimized systems would require 20 to 30 minutes to generate feature range rankings.
- Not only does STAR1 search faster but it explores a larger space. As far as we know, this is the first report of an extensive exploration of the space of tunings internal to COCOMO family prediction models.

[3]Wallace and Fujii's definitions of V&V [29]; the IEEE 2004 standard on V&V [16]; NASA's recommended V&V practices [17]

[4]E.g. improving software maturity decreases rework [19]

[5]`http://www.icsp-conferences.org/icsp2007/`

8. DISCUSSION

We defined "new ASE tools" to be:

$$\textit{automated analysis} \in \{5,6\} \vee$$
$$\textit{execution-based testing and tools} \in \{5,6\}$$

Making the business case for new ASE tools is complicated by an anti-automation bias ("sociology beats technology") and insufficient data for local tuning of prediction models.

We hypothesized that precise local tunings are not required to assess the relative merits of new ASE tools over other methods The space of tunings is well known- we just need to find what conclusions are stable within that space. Simulated annealing was used to sample that space and generate a set of 10% *best* and 90% *rest* solutions The N ranges in those solutions were ranked using a support based Bayesian method. Experiments were performed where the top $X \leq N$ ranged ranges were set and COCOMO, COQUALMO, and THREAT were allowed to randomly select the remaining ranges.

By varying the size of X, it was possible to find project decisions that most minimized effort/ defect/ threat. Inspecting those decisions for two NASA systems, we found that new ASE tools were optional in one (OSP) and essential in the other (OSP2).

Any generalization based on just two case studies should be treated with caution. The follow observations should therefore be checked on other projects:

- Recall that (a) OSP was an early prototype for OSP2 and that (b) new ASE tools were optional for OSP but essential for OSP2. That is, *project maturity may be a selector for the successful use of new ASE tools.*
- In Figure 8 and Figure 9, only very high and extremely high range automated analysis or execution-based testing and tools were used to minimize effort or defects or threats. Lower range usage of these tools was never useful. That is, when using new ASE tools, *use them thoroughly or not at all*.
- In our case studies augmenting sociological decisions with new ASE tools can lead to the greatest reduction in effort/ defects/ threats. That is *sociology does <u>not</u> beat technology* and *technology can compliment sociology*.
- When this study could make the business case for new ASE tools, a pre-condition for that case was the inclusion of sociological decisions (e.g. about process maturity and schedule pressure) along with the new ASE tools. Hence *ASE researchers need to study both software development technology <u>and</u> sociology*.

Our conclusions are project-specific but the method of generating them is general to any project that can be described in terms of the Figure 2 table. An appendix of this paper describes how to download, install, and use STAR1 to find the most important feature ranges that reduce effort / defects / threats for projects.

9. REFERENCES

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

[2] 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.

[3] 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.

[4] 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.

[5] J. Clark, J. J. Dolado, M. Harman, R. M. Hierons, B. Jones, M. Lumkin, B. Mitchell, S. Mancoridis, K. Rees, M. Roper, and M. Shepperd. Reformulating software engineering as a search problem. *IEE Proceedings on Software*, 150(3):161–175, 2003. Available from http://www.brunel.ac.uk/~csstrmh/papers/sbse.ps.

[6] R. Clark. Faster treatment learning, Computer Science, Portland State University. Master's thesis, 2005.

[7] S. L. Cornford, M. S. Feather, J. Dunphy, J. Salcedo, and T. Menzies. Optimizing spacecraft design optimization engine development: Progress and plans. In *Proceedings of the IEEE Aerospace Conference, Big Sky, Montana*, 2003. Available from http://menzies.us/pdf/03aero.pdf.

[8] T. DeMarco and T. Lister. *Peopleware: productive projects and teams*. Dorset House Publishing Co., Inc., New York, NY, USA, 1987.

[9] S. Devnani-Chulani. *Bayesian Analysis of Software Cost and Quality Models*. PhD thesis, 1999. Available on-line at http://citeseer.ist.psu.edu/devnani-chulani99bayesian.html.

[10] P. Domingos and M. J. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997.

[11] M. Feather and T. Menzies. Converging on the optimal attainment of requirements. In *IEEE Joint Conference On Requirements Engineering ICRE'02 and RE'02, 9-13th September, University of Essen, Germany*, 2002. Available from http://menzies.us/pdf/02re02.pdf.

[12] M. Hall and G. Holmes. Benchmarking attribute selection techniques for discrete class data mining. *IEEE Transactions On Knowledge And Data Engineering*, 15(6):1437– 1447, 2003. Available from http://www.cs.waikato.ac.nz/~mhall/HallHolmesTKDE.pdf.

[13] 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.

[14] K. Havelund and T. Pressburger. Model checking java programs using java pathfinder. *International Journal on Software Tools for Technology Transfer*, 2(4), April 2000. Available from http://ase.arc.nasa.gov/visser/jpf/jpf1.ps.gz.

[15] G. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.

[16] IEEE-1012. IEEE standard 1012-2004 for software verification and validation, 1998.

[17] B. Jackson, J. Griggs, K. Costello, and D. Solomon. Systems level definition of iv&v, 2006. NASA document IVV 09-1, last revised March 16, 2006, available on-line at http://www.nasa.gov/centers/ivv/pdf/170825main_IVV_09-1.pdf.

[18] J. Josephson, B. Chandrasekaran, M. Carroll, N. Iyer, B. Wasacz, and G. Rizzoni. Exploration of large design spaces: an architecture and preliminary results. In *AAAI '98*, 1998. Available from http://www.cis.ohio-state.edu/~jj/Explore.ps.

[19] J. King and M. Diaz. How CMM impacts quality, productivity, rework, and the bottom line. *CROSSTALK*, March 2002. Avaiable from http://www.stsc.hill.af.mil/crosstalk/2002/03/diaz.html.

[20] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[21] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[22] 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.

[23] T. Menzies and Y. Hu. Data mining for very busy people. In *IEEE Computer*, November 2003. Available from http://menzies.us/pdf/03tar2.pdf.

[24] T. Menzies, D. Raffo, S. on Setamanit, Y. Hu, and S. Tootoonian. Model-based tests of truisms. In *Proceedings of IEEE ASE 2002*, 2002. Available from http://menzies.us/pdf/02truisms.pdf.

[25] T. Menzies and J. Richardson. Xomo: Understanding development options for autonomy. In *COCOMO forum, 2005*, 2005. Available from http://menzies.us/pdf/05xomo_cocomo_forum.pdf. For more details, see also the longer technical report http://menzies.us/pdf/05xomo101.pdf.

[26] T. Menzies and E. Sinsel. Practical large scale what-if queries: Case studies with software risk assessment. In *Proceedings ASE 2000*, 2000. Available from http://menzies.us/pdf/00ase.pdf.

[27] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. *J. Chem. Phys*, 21:1087–1092, 1953.

[28] R. Rubinstein and D. Kroese. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*. Springer-Verlag, 2004.

[29] D. Wallace and R. Fujii. Software verification and validation: An overview. *IEEE Software*, pages 10–17, May 1989.

APPENDIX

A. OBTAINING AND USING STAR1

These instructions should support a LINUX and CYGWIN install of STAR1. In the event of technical difficulties, please contact the first two authors (Menzies or Elrawas).

```
bash -i
[ ! -d "$HOME/bin" ] && mkdir $HOME/bin
export PATH="$PATH:$HOME/bin"
wget http://unbox.org/wisp/tags/STAR/1.0/STAR_1.0.zip
unzip STAR_1.0.zip
make  # requires gcc
cd ~/STAR
cd eg
./1  | tee osp2.out | less
./2  | tee osp.out  | less
```

To run STAR1 on projects other than OSP or OSP2:

- Copy and edit one of the projects files in `~/STAR/STAR_projects`.
- Note that you will have to edit *both* a x.ranges and x.values file.
- Then copy and edit (e.g.) `~/STAR/eg/1` to point to your edited project details.