Learning Changes to Software Projects

Tim Menzies, *Member, IEEE*, Adam Brady,Jacky W. Keung, *Member, IEEE*,
Steven Williams, Oussama El-Rawas, Phillip Green

Abstract—
BACKGROUND: Given many possible changes to a software project, which ones are recommended?
AIM: To comparatively assess different decision procedures for recommending project changes.
METHOD: We search for project recommendations within data from eight projects using various AI tools: six model-based methods and one instance-based method called $\mathcal{W}2$. Results were assessed by comparing effort, defects, development time values in the raw data versus the subset of the data selected by those recommendations.
RESULTS: In the majority case, significantly large reductions on effort, defects and development time were achieved. Further, $\mathcal{W}2$ performed as well, or better, than anything else in this study. $\mathcal{W}2$ does not rely on an underlying model of software process so it does not demand that domain data be expressed in the terminology of that model. Hence, it can be quickly adapted to a new domain (just change the library of training instances) and easy to maintain (just add more instances).
CONCLUSION: We recommend instance-based methods like $\mathcal{W}2$ for learning changes to a software project.

Index Terms—Search-based software engineering, analogy, COCOMO

To attain knowledge, add things every day.
To attain wisdom, remove things every day.
– Lao-tse

1 INTRODUCTION

There are many *technologies* that a manager might apply to *change*, and hopefully *improve*, their software development project. Some technologies are *paper-based* methods like the checklists proposed by orthogonal defect classification [1]. Other technologies are *tool-based* such as using the new generation of functional programming languages or execution and testing tools [2] or automated formal analysis [3]. Yet other technologies are more *process-based* including process improvement initiatives, changing an organization's hiring practices, or a continual renegotiation of the requirements as part of an agile software development cycle [4].

The set of possible changes to a project is very long. Endres & Rombach [5] list a hundred *laws* of software engineering (each law predicts a certain observation) that could be used to justify a particular change to a project. Table 2 of the IEEE-1012 standard for software V&V offers a "minimal list" of 52 tasks [6]. If a manager proposed applying *all* the laws and *all* the V&V tasks, then their senior management would most probably ask them to scale back their plans to just a minimal set of most cost-effective measures.

- *Tim Menzies (corresponding author) Adam Brady, Phillip Green and Oussama El-Rawas are with the Lane Department of Computer Science and Electrical Engineering, West Virginia University. E-mail: tim@menzies.us, adam.m.brady@gmail.com, deathcheese@yahoo.com, orawas@gmail.com*
- *Jacky W. Keung is with the School of Computer Science and Engineering, University of New South Wales. E-mail: jacky.keung@nicta.com.au.*
- *Steven Williams is with the School of Informatics and Computing Indiana University, Bloomington. E-mail: stevencwilliams@gmail.com.*

This research is funded in part by NSF,CISE, project #0810879.

How to find the minimal set of useful changes to a project? It is ill-advised to just apply standard truisms since general principles may not work best for particular projects:

- In [7], we found one project where allocating extra resources to defect removal early in the life cycle (as recommended by Boehm [8]) was *less effective* than quickly deploying an initial working system, then fixing the bugs identified by users of that system.
- Elsewhere [9], we list numerous examples were applying general principles to a project were less valuable than applying specific changes selected for particular projects.

If locally derived changes are better than general truisms, then the research question becomes *what techniques can find useful changes for a particular project*. In this paper, we distinguish two approaches *model-based* and *instance-based* methods. Model-based methods build a model via expert advice [10] or some automatic method like data mining [11]. Once the model is built, it can be used for "what-if" queries in order to assess possible changes to a project.

$$\begin{array}{rcl} data & \rightarrow & model \\ model + whatIf & \rightarrow & predictions \end{array} \quad (1)$$

Instance-based methods insert the "what-if" query into a n-dimensional space populated with historical cases [12]–[16]. The immediate neighborhood of the "what-if" is surveyed and the prediction is some summary of those neighboring cases.

$$\begin{array}{rcl} data + whatIf & \rightarrow & neighborhood \\ neighborhood & \rightarrow & predictions \end{array} \quad (2)$$

Note that, unlike model-based methods, this instance-based approach makes no use of an underlying model.

Previously we have studied model-based methods [9], [17]–[24]. AI tools were used to control thousands of "what-if" queries over COCOMO models to find recommendations that reduced defects and/or months and/or effort.

Initially, the goal was a generic tool which could easily accommodate different AI tools and different models. This proved surprisingly difficult. Our model-based methods have become so intricate that subsequent modifications required elaborate background knowledge and prolonged programming.

When tools become complex, it is wise to ask if the complexity is *essential* or *superfluous*. Researchers into empirical methods such as Cohen [25] advise benchmarking supposedly more sophisticated method against a simpler alternative. Such "straw-man baseline" studies can yeild surprising results. For example, Holte reports that for standard data sets used in the machine learning literature, one-level decision trees work nearly as well as multiple-level decision trees [26]. Similar results are reported by Domingos & Pazzani for so-called Naive Bayes classifiers. Such classifiers ignore correlations between variables. Yet, surprisingly, they often perform as well as more complex learners (see Table 1 of [27]).

Accordingly, we conducted a straw-man baseline study of our AI model-based methods against a very simple instance-based approach called $\mathcal{W}$. Given a "what-if" query that selects for some set of similar projects, $\mathcal{W}$ seeks a $treatment$ R_x which finds the "better" parts of those similar projects:

$$\begin{array}{rcl} data + whatIf & \rightarrow & neighborhood_1 \\ neighborhood_1 & \rightarrow & predictions_1 \\ & & \\ R_x(neighborhood_1) & \rightarrow & neighborhood_2 \\ neighborhood_2 & \rightarrow & predictions_2 \\ value(predictions_2) & > & value(predictions_1) \end{array} \quad (3)$$

Here $value$ represents business concerns; e.g. reduce defects.

$\mathcal{W}$ finds the treatment R_x via a linear-time greedy search over ranges sorted by simple Bayesian ranking. The expectation was that $\mathcal{W}$ would be *too* simple and would perform poorly. However, when compared to model-based methods:

- $\mathcal{W}$ found *similar* or *better* treatments.
- $\mathcal{W}$ *was faster to run:* all the experiments of this paper take 10 minutes with $\mathcal{W}2$, but days for using models.
- $\mathcal{W}$ *was simpler to implement:* $\mathcal{W}2$'s 200 lines AWK replaces thousands of lines of the model-based LISP.
- $\mathcal{W}$ *was simpler to maintain:* with instance-based methods, "maintenance" means just "add more instances".
- $\mathcal{W}$ *was simpler to adapt to new domains:* $\mathcal{W}$ makes no use of an underlying model and is therefore imposes no restrictions on the data being processed. Hence it can be quickly applied to more data sets.

This last point is most important. Model-based tools only accept data that conforms to the ontology of the model (i.e. use the input values of the model). If local data does not conform to that ontology, then the tool cannot be applied. Also, the conclusions of a model-based analysis are only as good as the underlying model. Models are learned from historical data and if that data is not relevant to some new project, then that model will be inappropriate for the new project. For example, Jorgensen & Shepperd caution that the data used to derive COCOMO may not apply to all corporations [28]. Since $\mathcal{W}$ does not use models, it avoids both these problems.

The contribution of this work is three-fold. Firstly, we demonstrate the practicality of learning changes to software projects that select for higher quality measures:

Dataset	Cols	Rows	Notes	Quality Measures
Kemerer	7	15	Large business applications	effort
Telecom	3	18	U.K. telecom enhancements	effort
Finnish	8	38	Finnish IS projects	effort
Miyazaki	8	48	Japanese COBOL projects	effort
NASA93	26	93	NASA projects	effort, time, defects
COC81	26	63	NASA projects	effort, time, defects
China	17	499	Chinese software projects	effort
		Total: 774		

Fig. 1. Seven data sets from promisedata.org/?cat=14: *effort* is total staff months; *time* is calendar time (start to stop); *defects* is number of delivered defects.

- In six case studies, we show that it is possible to find changes to projects that reduce the development effort;
- In two other case studies that it is possible to find changes to projects that reduce development effort *and* delivered defects *and* the development time.

Secondly, we show that a very simple instance-based method does this better than more complex model-based methods.

Thirdly, this work offers the software engineering community a cautionary tale. Decades of research has offered us any number of ways to analyze project data. For example, consider the list of candidate methods that might be applied to the problem of learning changes to software projects. From the data mining [29], AI [12], [30], [31], constraint-satisfaction [32], and search-based SE literature [33] we see that that list includes simulated annealing; evolutionary algorithms; tabu search; ant-based search; various data mining tools such as support vector machines or random forests; various Davis-Putnam procedures; logic based-approaches such as binary-decision diagrams; numeric methods such as integer programming; or any number of AI search algorithms. This list is hardly exhaustive (no doubt, the reader can add their own favorite analysis method).

Given such a large and growing menagerie of candidate tools, the temptation is to always explore novel and more intricate tools. Certainly, some domains are inherently complex and should be analyzed with complex tools. For example, *next release planning* is a complex planning problem requiring intricate optimization methods [34], [35].

On the other hand, most of the data sets explored by (say) the effort estimation literature are not complex:

- The effort estimation datasets used in Mendes et al. [36], Auer et al. [37], Baker [38], and Li et al. [39] have a median number of rows 13,15,33,52 (respectively).
- Figure 1 shows the data explored in this study. While one of our data sets (China) is moderately larger, most of them contain just a few dozen rows or less.

For such small search spaces, complex tools may be an overkill. To avoid superfluous complexity, researchers should apply Cohen's test; i.e. benchmark their supposedly better tool against a simpler tool. For learning changes to software projects, $\mathcal{W}$ is a candidate tool for such a comparison since:

- It is simple to implement and fast to execute;
- The algorithm scales well (runs in linear time);
- For the data studied here, it performs as least as good (or better) than a range of model-based methods.

The rest of this paper presents our model-based vs instance-based comparison. First, our model-based results [9], [17]–[24] are reviewed. That work resulted in SEESAW, our "best" AI tool for conducting "what-if" model-based queries for software process models. Next, we describe $\mathcal{W}2$, an instance-based method that improves on prior versions of $\mathcal{W}$ [40], [41]. This is followed a comparison of model- vs instance-based methods for learning software project changes.

2 MODEL-BASED METHODS

The task of optimizing for a set of goals is traditionally solved by computing partial differential equations of a model, then exploring the surface of steepest change. A premise of this approach is *tuning stability*; i.e. that the gradients at any point in the model can be determined with certainty. For some models, this premise does not hold. Consider the following simplified COCOMO [42] model:

$$effort = a \cdot LOC^{b+pmat} \cdot acap \tag{4}$$

While simplified, the equation presents COCOMO's core assumption that software development effort is exponential on program size. In this equation, (a, b) control the linear and exponential effects (respectively) on model estimates; while $pmat$ (process maturity) and $acap$ (analyst capability) are project choices adjusted by managers. Equation 4 contains two features ($acap, pmat$) and a full COCOMO model contains many more project descriptors (e.g. see Figure 2).

Baker [38] learned values of (a, b) for a full COCOMO model using Boehm's local calibration method [43] from 300 random samples of 90% of the available project data. As shown in Figure 3, the ranges varied widely and wildly:

$$(3.2 \leq a \leq 9.4) \wedge (0.8 \leq b \leq 1.12)$$

Such large variations confuse standard gradient descent methods as well as obscuring the effects of project changes. Suppose some proposed technology doubles productivity, but a moves from 9 to 4.5. The improvement resulting from that change would be obscured by the tuning instability.

If tuning instability cannot be removed, it must be managed. Our model-based methods assume that model predictions are altered by project variables P and tuning variables T:

$$prediction = model(P, T) \tag{5}$$

E.g. in local calibration, the tuning options T are the ranges of (a, b) while the project options P are the EM_i values.

At any local site, only part of the tunings is relevant: we denote these as $t \subseteq T$. This subset can be found in many ways including linear regression or local calibration. However, if there is insufficient data for stable tunings, then T may as well be left unconstrained, so $t \subseteq T$ can be selected randomly.

Managers explore a specific context (the particulars of their project) $context \subseteq P$ and $control$ some items of $context$ ($control \subseteq context$). Since it is too expensive to use all $control$ settings, we seek minimal treatments $R_x \subseteq control$; i.e. no smaller treatment has the same (or better) effect as R_x.

upper:	acap: analyst capability
in theory	apex: applications experience
$\beta < 0$	ltex: language and tool-set experience
	pcap: programmer capability
	pcon: personnel continuity
	plex: platform experience
	site: multi-side development
	tool: use of software tools
middle	sced: dictated development scedule
lower:	cplx: product complexity
in theory	data: database size
$\beta > 0$	docu: documentation
	pvol: platform volatility
	rely: required reliability
	ruse: required reuse
	stor: required % of available RAM
	time: required % of available CPU

Fig. 2. COCOMO II effort multipliers.

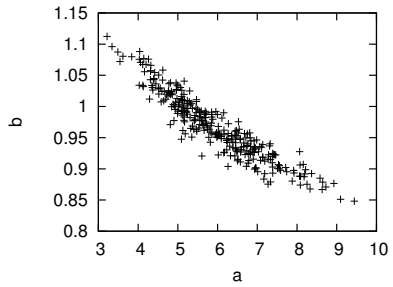


Fig. 3. Learned (a, b) values via local calibration on 300*90% samples of the NASA93 COCOMO data set.

Models assess different treatments by running them on the model and returning the one that maximizes a model's $value$:

$$\overbrace{R_x \subseteq control}^{\text{AI search}}, median_n \left(\underbrace{t \subseteq T, value(model(R_x, t))}_{\text{random selection}} \right) \tag{6}$$

Here, $median_n$ reports the median seen in n repeats of the random selection and *value* is a domain-specific function For example, $value$ could be computed from the difference of the model estimates to zero (effort, defects, development time):

$$value = 1 - \left(\sqrt{Effort^2 + Defects^2 + Time^2} / \sqrt{3} \right) \tag{7}$$

If the estimates are normalized to the range $0 \ldots 1$, then $0 \leq value \leq 1$ and *higher* values are *better*.

2.1 Tuning Variance in COCOMO

Equation 6 requires that we sample over the space of model tunings. This section describes one mechanism for sampling across the space of the tunings within Boehm's COCOMO effort estimator and the COQUALMO defect estimator. Perhaps the essential point of this section is that, for instance-based methods, *none* of this machinery is required.

For COCOMO effort multipliers (the features that that affect effort/cost in a linear manner), the off-nominal ranges {vl=1, l=2, h=4, vh=5, xh=6} change the prediction by some ratio. The nominal range {n=3} corresponds to an effort multiplier of 1 (i.e. no change). 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:

$$\forall x \in \{1..6\}\ EM_i = m_\alpha(x - 3) + 1 \tag{8}$$

where m_α is the effect of α on effort/cost. The exponential influences on cost/effort take the form:

$$\forall x \in \{1..6\}\ SF_i = m_\beta(x - 6) \tag{9}$$

where m_β is the effect of factor i on effort/cost.

COQUALMO contains equations of the same syntactic form as Equation 8 and Equation 9, but with different coefficients. Using experience for 161 projects [42], we can find the maximum and minimum values ever assigned to m for COQUALMO and COCOMO. Hence, to explore tuning variance (the $t \in T$ term in Equation 6), all we need to do is select m values at random from the min/max m values ever seen.

In the following, m_α, m_β are COCOMO's linear,exponential influences on effort and cost, and m_γ, m_δ are COQUALMO's linear,exponential influences on defects.

There are two sets of effort/cost multipliers. The *positive effort EM* features, with slopes m_α^+, that are proportional to effort/cost. These features are: cplx, data, docu, pvol, rely, ruse, stor, and time. The *negative effort EM* features, with slopes m_α^-, are inversely proportional to effort/cost. These features are acap, apex, ltex, pcap, pcon, plex, sced, site, and tool. The m ranges, as seen in 161 projects [44], are:

$$\left(0.073 \le m_\alpha^+ \le 0.21\right) \wedge \left(-0.178 \le m_\alpha^- \le -0.078\right) \tag{10}$$

In the same sample of projects, the COCOMO effort/cost scale factors (prec, flex, resl, team, pmat) have the range:

$$-1.56 \le m_\beta \le -1.014 \tag{11}$$

There are two sets of defect multipliers and scale factors. The *positive defect* features have slopes m_γ^+ and are proportional to estimated defects. These features are flex, data, ruse, cplx, time, stor, and pvol. 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.

COQUALMO divides describes how defects change in requirements, design, and coding with tuning options:

$$\begin{aligned} requirements &\left\{ \begin{array}{c} 0 \le m_\gamma^+ \le 0.112 \\ -0.183 \le m_\gamma^- \le -0.035 \end{array} \right. \\ design &\left\{ \begin{array}{c} 0 \le m_\gamma^+ \le 0.14 \\ -0.208 \le m_\gamma^- \le -0.048 \end{array} \right. \\ coding &\left\{ \begin{array}{c} 0 \le m_\gamma^+ \le 0.14 \\ -0.19 \le m_\gamma^- \le -0.053 \end{array} \right. \end{aligned} \tag{12}$$

The tuning options for the defect removal features are:

$$\begin{array}{rl} \forall x \in \{1..6\} & SF_i = m_\delta(x - 1) \\ requirements: & 0.08 \le m_\delta \le 0.14 \\ design: & 0.1 \le m_\delta \le 0.156 \\ coding: & 0.11 \le m_\delta \le 0.176 \end{array} \tag{13}$$

where m_δ denotes the effect of i on defect removal.

	context				
	$control$able			un$control$able	
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		
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		
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 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		

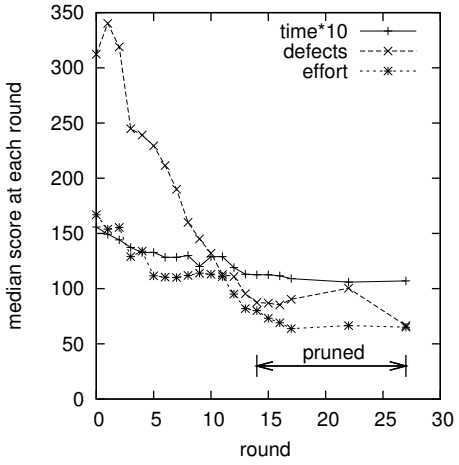
Fig. 4. Contexts of 4 case studies. {*1, 2, 3, 4, 5, 6*} map to {*very low, low, nominal, high, very high, extra high*}.

2.2 Case Studies

Figure 4 shows some real-world $context$ and $control$ information taken from a debrief of some NASA program managers:

- *Ground* and *flight* represent typical ranges for most NASA projects at the Jet Propulsion Laboratory (JPL);
- *OSP* represents the guidance, navigation, and control aspects of NASA's 1990 Orbital Space Plane;
- *OSP2* represents a second, later version of OSP with a more limited scope of COCOMO attributes.

The un$control$able column in that figure shows project features that cannot be changed; e.g. for OSP, the required reliability is fixed at $rely = 5$. On the other hand, the low and $high$ ranges in that figure define the space of possible changes to that project. For instance, the reliability of the JPL



Decisions made from round=1 to round=13:

x=0: $R_x = \emptyset$	x=7: added {rely=3}
x=1 added {pmat=3}	x=8: added {stor = 3}
x=2: added {resl=4}	x=9: added {time = 3}
x=3: added {team=5}	x=10: added {tool = 4}
x=4: added {aexp=4}	x=11: added {sced = 2}
x=5: added {docu=3}	x=12: added {site = 4}
x=6: added {plex=4}	x=13: added {acap = 5}

Fig. 5. Example of SA's forward and back select.

flight software can vary from a ranking of 3 (nominal) to 5 (very high).

2.3 Six AI Model-Based Algorithms

The case studies of Figure 4 can be used to assess how well different AI algorithms can find changes to software projects. For example, a typical simulated annealing [45] (SA) run explores 10,000 variants on some solution. A side-effect of that run is 10,000 sets of inputs, each scored with the $value$ function of Equation 7. Our tool classified the outputs into the 90% *rest* and the 10% *best* seen during the run of the SA. All the ranges from all the features were then ranked according to how much more frequently they appeared in *best* than *rest*. A *forward select* was then called using the first $1 \ldots x$ ranked items. For example, in Figure 5, the treatment R_x at any x value is the conjunction of ranges seen one to x (see the table at the bottom of that figure). The y axis scores shows median results in 100 runs of COCOMO/COQUALMO, after imposing the treatment. The "pruned" range of that figure shows the results of a *back select* that worked backwards over the forward select ordering, deleting any item x whose distribution of values was statistically insignificantly different to $x - 1$. SA's final recommendation was the treatment $1 \leq x \leq 13$. The $improvement$ generated by that treatment can be seen by comparing the values at $x = 0$ to $x = 13$.

- Defects reduced: 350 to 75;
- Time reduced: 16 to 10 months;
- Effort reduced: 170 to 80 staff months.

SA is just one way to generate a treatment. For our AI model-based methods, we explored five others. Given a random selected treatment, *MaxWalkSat* tries n modifications to randomly selected features [46]. Sometimes (controlled by the α parameter), the algorithm chooses the range that minimizes the value of the current solution. Other times (at probability $1 - \alpha$), a random range is chosen for the feature. After N retries, the best solution is returned. Our implementation used $n = 50$, $\alpha = 0.5$, and $N = 10$.

SEESAW [22] augments MaxWalkSat with a search heuristic taken from simplex optimization. SEESAW ignores all ranges except the minimum and maximum values for a feature in p. Like MaxWalkSat, the feature chosen on each iteration is made randomly. However, SEESAW has the ability to delay bad decisions until the end of the algorithm (i.e., decisions where constraining the feature to *either* the minimum or maximum value results in a worse solution). Hence, SEESAW's search was followed by the same back-select process using for SA.

ISAMP is a fast stochastic iterative sampling method that extends a treatment using randomly selected ranges. The algorithm follows one solution, then resets to try other paths (our implementation resets 20 times). The algorithm has proved remarkably effective at scheduling problems, perhaps because it can rapidly explore more of the search space [47]. To avoid exploring low-value regions, our version of ISAMP stores the worst solution seen so far. Any conjunction whose $value$ exceeds that of the worst solution is abandoned, and the new "worst value" is stored. If a conjunction runs out of new ranges to add, then the "worst value" is slightly decreased. This ensures that consecutive failing searches do not permanently raise the "worst value" by an overly permissive amount.

Our remaining algorithms use some variant of tree search. Each branch of the tree is a different "what-if" query of size i. If i is less than the number of input values to COCOMO/COQUALMO, the missing values were selected at random from the legal ranges of those inputs.

BEAM search extends search branches as follows. Each branch forks once for every new option available to that range. All the new leaves are sorted by their value and only the top N ranked branches are marked for further expansion. For this study we used $N = 10$ and results scored using the median $values$ seen in the top N branches.

A-STAR runs like BEAM, but the sort order is determined by the sum f (the cost of reaching the current solution) plus g (a heuristic estimate of the cost to reach the final solution). Also, unlike BEAM, the list of options is not truncated so a termination criterion is needed (we stop the search if the best solution so far has not improved after m iterations). For this study, we estimated f and g as follows:

- f was estimated as the percentage of the project descriptors with ranges in the current branch;
- g was estimated using $1 - Equation\ 7$ (i.e. distance to the utopia of no effort, no development time, and no defects).

Initially, we planned to explore more methods than SA, MaxWalkSat, SEESAW, ISAMP, BEAM and A-STAR. However, the success of $\mathcal{W}$'s instance-based approach decreased our motivation to explore other AI model-based methods.

2.4 Comparisons of AI Model-based Methods

The above AI algorithms implemented the "AI search" of Equation 6 to explore the tuning variance of the models in

algorithm	Defects	months	time
SEESAW	4	4	3
BEAM	0	3	3
A-star	0	1	1
SA	0	1	1
MaxWalkSat	0	0	0
ISAMP	0	0	0

Fig. 6. Number of times algorithm were top-ranked (largest is 4: i.e. one for each Figure 4 case study).

§2.1 (Boehm's COCOMO effort end time estimator and his COQUALMO defect predictor). In four case studies, $context$ was set to the case studies of Figure 1.

Each algorithm was run 20 times and was guided by the $value$ function of Equation 7. Separate statistics were collected for the defects/effort/time predictions seen at the policy point in the 20*4 trials. The *top-ranked* algorithm(s) of Figure 6 had statistically different and lower defects/effort/time predictions than any other algorithm(s).

Note the dramatic difference between MaxWalkSat and SEESAW results. The difference between these two algorithms is very small: SEESAW assumed that the local search state space was monotonic, so it only explored minimum and maximum values for each feature. This result underscores the power of the simplex heuristic.

From Figure 6, the worst algorithms are MaxWalkSat and ISAMP and the best algorithms are SEESAW and BEAM. The performance of these best algorithms is sometimes equivalent (e.g., in $time$, both algorithms achieved an equal number of top ranks). However, BEAM is not recommended:

- BEAM runs 10 times slower than SEESAW.
- SEESAW performs better than BEAM in some cases (e.g. in defects, BEAM is never top-ranked).

Since SEESAW performs best, we will use it for our subsequent comparisons with instance-based methods.

3 INSTANCE-BASED METHODS

This section described three version of the $\mathcal{W}$ instance-based tool. Lessons learned from $\mathcal{W}0$ [40] and $\mathcal{W}1$ [41] will inform the description of the current version, $\mathcal{W}2$.

Like all instance-based methods, $\mathcal{W}$ assumes access to historical $cases$ described using P project descriptors (e.g. analyst capability; process maturity; etc). Note that, unlike the model-based approach, $\mathcal{W}$ does *not* assume that all cases are described using the same set of P descriptors. Rather, P can be varied. For example, Figure 1 lists the data sets used in our analysis. If $\mathcal{W}$ was restricted to just the COCOMO ontology, then it could only analyze two of those seven data sets: NASA93 and COCOMO81. $\mathcal{W}$'s applicability to a wide range of data sets is an important advantage over the AI model-based methods described above.

$\mathcal{W}$'s next assumption is that each case is described by a set of qualities such as number of defects, development time, total staff effort etc. All these qualities are summarized into a single value by some $value$ function like Equation 7.

Like our model-based methods, $\mathcal{W}$ assumes that a manager can offer us: a description of the $context \subseteq P$ that interests them and a list of $control$able options which they can change ($control \subseteq context$). For example, once we asked a NASA software project manager for a description of the effects assigning inexperienced people. The manager commented that, at her site, such inexperience implies low applications experience ($aexp$), low to very low platform experience ($plex$), and language and tool experience ($ltex$) that is not high. Next, we asked the manager to describe the range of projects seen at his site (using the COCOMO names of Figure 2). The resulting $context_1$ is shown below:

$$
\begin{aligned}
context_1 = \\
&apex \in \{2\} \wedge plex \in \{1,2\} \wedge ltex \in \{1,2,3\} \wedge \\
&?pmat \in \{2,3\} \wedge ?rely \in \{3,4,5\} \wedge ?data \in \{2,3\} \wedge \\
&?cplx \in \{4,5\} \wedge ?time \in \{4,5\} \wedge ?stor \in \{3,4,5\} \wedge \\
&?pvol \in \{2,3,4\} \wedge ?acap \in \{3,4,5\} \wedge ?pcap \in \{3,4,5\} \wedge \\
&?tool \in \{3,4\} \wedge ?sced \in \{2,3\}
\end{aligned}
$$

Here, "?" are the $control$lables; for example, this manager is senior enough adjust factors like schedule pressure ($sced$).

Note that there is no requirement for managers to include all project descriptors in their $context$ statement. As seen below, $\mathcal{W}$ can handle contexts that are a subset of the descriptors.

$\mathcal{W}$ finds a project treatment R_x by studying the project similar to the context in the case library. Formally, $\mathcal{W}$ explores the $neighorhood$ of the $context$, looking for ways to select for the "best" cases. (as determined by a $value$ functions like Equation 7). In $\mathcal{W}2$, this is a six step procedure:

1) Divide cases randomly into $train : test$ in the ratio 2:1.
2) Used $context$ to find the neighborhood within $train$.
3) Divide neighborhood into (a) the $best$ cases that should be emulated, and (b) the remaining cases you should avoid (which we call $rest$).
4) Rank all differences between (a) and (b) according to how strongly they select for the $best$ cases.
5) Using the $train$ set again, experiment with treatments R_x built from the top ranked items found in $Step4$. Return the treatment that selects for the cases in the $train$ set with highest median $value$.
6) Repeat five steps $n = 20$ times with other randomly selected $train : test$ sets. Prune $unstable$ treatments; i.e. those not found in the majority of repeats.

$\mathcal{W}0$ and $\mathcal{W}1$ neglected to test for treatment stability; hence, $Step6$ was added. $Step6$ is also useful to resolve another testing-related problem. Other learners such as RIPPER [48] divide the case data three ways into $train : prune : test$. In that approach, $Step4$ would run on $train$; $Step5$ would run on $prune$, and the final result would be tested on a separate $test$ set. $\mathcal{W}0$ tried that approach but ran into problems with very small data sets such as the 15 rows of KEMERER or the 18 rows of TELECOM. $\mathcal{W}2$ uses the above two-way split, and uses $Step6$ to avoid recommending treatments that are over-fitted to parts of the training data.

Another issue encountered with $Step2$ (finding the neighborhood) was that the $context$ cannot be treated as a rigid criteria. In our experiments with $\mathcal{W}0$, we found that some data sets were so small that, often, $none$ of the cases contained all the ranges mentioned in the $context$. For example, Figure 7.a shows training data from NASA93. The gray cells in that figure show ranges that do not appear in $context_1$. Note that

Figure 7.a: RELEVANT= cases nearest to $context_1$.

row	*apex*	*plex*	*ltex*	pmat	rely	data	cplx	time	stor	pvol	acap	pcap	tool	sced	effort	overlap
57	3	2	2	3	4	3	5	5	5	4	3	3	3	3	38	13
56	3	2	2	3	4	3	5	5	5	4	3	3	3	3	12	13
55	3	2	2	3	4	3	5	5	5	4	3	3	3	3	480	13
53	2	1	2	2	5	2	5	5	6	2	4	3	4	3	648	13
35	4	3	3	2	4	3	4	4	4	2	3	3	3	3	370	12
26	3	3	3	3	3	3	4	4	3	3	3	3	3	3	114	12
09	4	2	1	3	3	2	4	3	3	4	4	4	3	3	215	12
40	4	3	4	3	4	3	4	4	3	2	4	4	3	3	636	11
25	3	3	3	3	3	3	4	3	3	3	3	3	3	3	42	11
23	3	3	3	3	3	3	4	3	3	3	3	3	3	3	60	11
22	3	3	3	3	4	3	4	3	3	3	3	3	3	3	42	11
17	4	3	3	3	4	3	4	3	3	3	3	4	3	3	210	11
16	4	3	3	3	4	3	3	4	3	3	3	4	3	3	90	11
47	3	4	4	4	4	3	5	4	4	2	4	3	3	3	703	10
44	4	4	4	2	3	3	4	3	5	2	4	4	3	2	300	10
43	4	4	4	2	3	3	4	3	5	2	4	4	3	2	300	10
41	4	4	4	2	4	3	4	3	5	2	4	4	3	2	576	10
36	3	2	3	4	3	4	5	3	3	2	4	5	3	2	278	10
34	4	3	4	2	3	4	4	5	3	3	4	4	3	3	155	10
33	4	3	4	2	3	4	4	5	3	3	4	4	3	3	98.8	10

(other cases omitted)

Figure 7.b: $Best$ (top) & $rest$ (bottom).

row	*apex*	*plex*	*ltex*	pmat	rely	data	cplx	time	stor	pvol	acap	pcap	tool	sced	effort
56	3	2	2	3	4	3	5	5	5	4	3	3	3	3	12
08	5	3	2	3	3	2	4	3	3	2	4	3	3	3	36
57	3	2	2	3	4	3	5	5	5	4	3	3	3	3	38
22	3	3	3	3	4	3	4	3	3	3	3	3	3	3	42
25	3	3	3	3	3	3	4	3	3	3	3	3	3	3	42
12	5	3	4	3	3	2	4	3	3	2	4	4	3	3	48
11	4	3	4	3	3	2	4	3	3	2	4	4	3	3	60
23	3	3	3	3	3	3	4	3	3	3	3	3	3	3	60
19	4	2	4	4	3	5	4	5	5	2	5	3	3	2	62
16	4	3	3	3	4	3	3	4	3	3	3	4	3	3	90
33	4	3	4	2	3	4	4	5	3	3	4	4	3	3	98.8
26	3	3	3	3	3	3	4	4	3	3	3	3	3	3	114
17	4	3	3	3	4	3	4	3	3	3	3	4	3	3	210
09	4	2	1	3	3	2	4	3	3	4	4	4	3	3	215
44	4	4	4	2	3	3	4	3	5	2	4	4	3	2	300
07	5	3	4	3	3	2	4	3	3	2	4	5	3	3	360
35	4	3	3	2	4	3	4	4	4	2	3	3	3	3	370
55	3	2	2	3	4	3	5	5	5	4	3	3	3	3	480
40	4	3	4	3	4	3	4	4	3	2	4	4	3	3	636
53	2	1	2	2	5	2	5	5	6	2	4	3	4	3	648

Figure 7.c: Rank with Equation 14.

range	frequency b best	frequency r rest	$b^2/(b+r)$
pmat=3	5/5	10/15	60%
sced=3	5/5	13/15	54%
tool=3	5/5	14/15	52%
acap=3	4/5	7/15	51%
data=3	4/5	9/15	46%
rely=4	3/5	6/15	36%
time=3	3/5	7/15	34%
pvol=4	2/5	2/15	30%
stor=3	3/5	10/15	28%
cplx=5	2/5	3/15	27%
stor=5	2/5	3/15	27%
cplx=4	3/5	12/15	26%
time=5	2/5	4/15	24%
pvol=3	2/5	5/15	22%
data=2	2/5	5/15	22%
rely=3	2/5	9/15	16%
pvol=2	1/5	9/15	5%

Fig. 7. Processing $context_1$ on a training set selected from NASA93.

all rows have at least one gray cell. That is, $none$ of that data exactly matches $context_1$. Consequently, $\mathcal{W}1$ used some partial match operator to compute the neighborhood. Members of the training set were sorted according to their Euclidean distance from the $context$. If a $context$ only mentions a subset of the project descriptors P, then $\mathcal{W}1$'s distance function filled in $\{P - context\}$ with random values (selected from the known ranges). This was repeated 50 times to generate 50 context queries that reference all project descriptors. Next, the neighborhood was computed using the intersection of the 20 instances closest to any of those 50 queries.

This technique solved the problem of missing parts of the $context$. However, it created other another problem: a large variance in all the results. Hence, $\mathcal{W}2$ used another method: $context$ became a set overlap membership function. For example, $context_1$ statement defines ranges for 14 project descriptors. Any training case contain ranges that overlaps with between 0 and 14 of the ranges in $context_1$. As shown in Figure 7.a, we can use the size of this overlap to sort the cases. Appealing to the central limit theorem, $\mathcal{W}2$ defines the neighborhood of a query to be the $K_1 = 20$ training cases with largest overlap to the query. If a $train$ set was smaller than 20 cases, all its rows were repeatedly copied until $|rows > 20|$.

$\mathcal{W}2$'s set overlap operator is only defined for finite ranges. Hence, $\mathcal{W}2$ adds a $Step0$: discretization of all project descriptors (but not quality attributes) into B bins. All the following experiments assume that all numbers y are discretized using $round((y - min)/((max - min)/B))$, where $B = 5$. Other values for B are discussed in the future work section.

$Step3$ (division into $best$ and $rest$) is illustrated in Figure 7.b. The neighborhood is sorted by case $value$; the top K_2 cases are $best$; and the remaining $K_1 - K_2$ examples are $rest$. While the ranking algorithm of $Step4$ works best for larger K_2 values, we did not want to exceed accepted standards in the research community. After a review of the analogy-based estimation literature [36], [39], [49]–[51] we noted that no researcher proposed using more than five neighbors. Hence, we used the $K_2 = 5$ cases with highest $value$ (in this example, $value$ means lower development effort).

$Step4$ (rank the ranges in $best$) is shown in Figure 7.c. $\mathcal{W}0$ used the following simple Bayesian ranking method. Observer that nominal tool ($tool = 3$) occurs 5 times in $best$ and 14 times in $rest$. Given that information, we can rank $tool = 3$ according to its ability to select $best$ cases:

$$\begin{aligned} E &= (tool = 3) \\ freq(E|best) &= 5 \\ freq(E|rest) &= 14 \\ ratio(E|best) &= 5/5 = 1 \\ ratio(E|rest) &= 10/15 = 0.93 \\ rank(E) &= \frac{ratio(E|best)}{ratio(E|best) + ratio(E|rest)} = 0.52 \end{aligned}$$

$\mathcal{W}0$ encountered problems with evidence that was infrequent, but relatively more frequent in $best$ than $rest$. To avoid this problem, $\mathcal{W}1$ and $\mathcal{W}2$ adds a support term. Support should $increase$ as the frequency of a range $increases$, i.e. $ratio(E|best)$ is a valid support measure. Hence, $\mathcal{W}2$'s range ranking formulae is:

$$rank(E) * support(E) = \frac{ratio(E|best)^2}{ratio(E|best) + ratio(E|rest)} \quad (14)$$

We can generate candidate treatments R_x by combining the top x items in the rankings of Figure 7.c:

$$\begin{aligned} R_1 &: pmat = 3 \\ R_2 &: pmat = 3 \wedge sced = 3 \\ R_3 &: pmat = 3 \wedge sced = 3 \wedge tool = 3 \\ R_4 &: pmat = 3 \wedge sced = 3 \wedge tool = 3 \wedge acap = 3 \\ etc & \end{aligned}$$

$Step5$ (pruning the treatments) applies R_x to the projects similar to the $context$; i.e. those found in the test set's neighborhood. For example, Figure 8 shows the $K_1 = 20$ cases closest to $context_1$ in the train set.

Figure 9 shows the cases from this neighborhood that satisfy $R_1 : pmat = 3$. It turns out, that for cases relevant to $context_1$ in this test set, there is some association between the ranges see in R_1, R_2 and R_3: all these treatments select the same

rows. Only when R_4 is applied does Figure 9 shrink to the two rows containing $acap = 3$. $\mathcal{W}1$'s results exhibited large variances if we drew conclusions from less than three rows. Hence, $\mathcal{W}2$'s explores R_x upwards from $x = 1$ while:

- The number of selected rows $|R_x \wedge neighborhood| \geq 3$;
- The median $value$ of the rows selected by R_{x+1} is greater than that of R_x.

In the case of Figure 9, $Step5$ returns R_1 ($pmat = 3$).

3.1 Measuring Performance

To compare the effectiveness of different treatments, we offer the following performance measures:

- All our measures are taken from the test set.
- The $baseline$ values are from the neighborhood of the $context$; e.g. the $effort$ column of Figure 8.
- The $treated$ values are from the cases selected by a treatment; e.g. the $effort$ column of Figure 9.
- The $median$ of a distribution is the 50-th percentile of the sorted values in that distribution.
- The $spread$ of a distribution is the (75-25)th percentile of the sorted values. We use spread rather than, say, standard deviation to avoid any parametric assumptions that our distributions are (say) Gaussian.
- The $improvement$ from $b = baseline$ to $t = treated$ is $100 * (b - t)/t$. $Larger$ improvements are better.

row	*apex*	*plex*	*ltex*	pmat	rely	data	cplx	time	stor	pvol	acap	pcap	tool	sced	effort	overlap
57	3	2	2	3	4	3	5	5	5	4	3	3	3	3	38	13
56	3	2	2	3	4	3	5	5	5	4	3	3	3	3	12	13
55	3	2	2	3	4	3	5	5	5	4	3	3	3	3	480	13
53	2	1	2	2	5	2	5	5	6	2	4	3	4	3	648	13
35	4	3	3	2	4	3	4	4	4	2	3	3	3	3	370	12
26	3	3	3	3	3	3	4	4	3	3	3	3	3	3	114	12
09	4	2	1	3	3	2	4	3	3	4	4	4	3	3	215	12
40	4	3	4	3	4	3	4	4	3	2	4	4	3	3	636	11
25	3	3	3	3	3	3	4	3	3	3	3	3	3	3	42	11
23	3	3	3	3	3	3	4	3	3	3	3	3	3	3	60	11
22	3	3	3	3	4	3	4	3	3	3	3	3	3	3	42	11
17	4	3	3	3	4	3	4	3	3	3	3	4	3	3	210	11
16	4	3	3	3	4	3	3	4	3	3	3	4	3	3	90	11
47	3	4	4	4	4	3	5	4	4	2	4	3	3	3	703	10
44	4	4	4	2	3	3	4	3	5	2	4	4	3	2	300	10
43	4	4	4	2	3	3	4	3	5	2	4	4	3	2	300	10
41	4	4	4	2	4	3	4	3	5	2	4	4	3	2	576	10
36	3	2	3	4	3	4	5	3	3	2	4	5	3	2	278	10
34	4	3	4	2	3	4	4	5	3	3	4	4	3	3	155	10
33	4	3	4	2	3	4	4	5	3	3	4	4	3	3	98.8	10

Fig. 8. The $K_1 = 20$ neighborhood of $context_1$ in a train set taken from NASA93.

row	*apex*	*plex*	*ltex*	pmat	rely	data	cplx	time	stor	pvol	acap	pcap	tool	sced	effort	overlap
57	3	2	2	3	4	3	5	5	5	4	3	3	3	3	38	13
56	3	2	2	3	4	3	5	5	5	4	3	3	3	3	12	13
55	3	2	2	3	4	3	5	5	5	4	3	3	3	3	480	13
26	3	3	3	3	3	3	4	4	3	3	3	3	3	3	114	12
09	4	2	1	3	3	2	4	3	3	4	4	4	3	3	215	12
40	4	3	4	3	4	3	4	4	3	2	4	4	3	3	636	11
25	3	3	3	3	3	3	4	3	3	3	3	3	3	3	42	11
23	3	3	3	3	3	3	4	3	3	3	3	3	3	3	60	11
22	3	3	3	3	4	3	4	3	3	3	3	3	3	3	42	11
17	4	3	3	3	4	3	4	3	3	3	3	4	3	3	210	11
16	4	3	3	3	4	3	3	4	3	3	3	4	3	3	90	11

Fig. 9. All rows of Figure 8 satisfying $R_1 : pmat = 3$.

For example, consider how $pmat = 3$ effects projects similar to $context_1$:

- Without the $pmat = 3$ restriction, the median and spread of the projects similar to $context_1$ are 235 and 633 months, respectively (see Figure 8).
- After imposing the $pmat = 3$ restriction, the median and spread of projects similar to $context_1$ are 81 and 353 months (see Figure 9).
- The observed improvement in the median is hence 66%.
- The observed improvement in the spread is hence 44%.

3.2 Complexity

Summarizing the above, $\mathcal{W}2$ studies c cases which describe projects using to $p = |P|$ project descriptors. Much of the processing is focused on the K_1 nearest neighbors to the $context$. $\mathcal{W}2$ repeats its process n times (to check for stability of the learned treatment). Prior to the repeats, any numeric descriptors are discretized into B bins.

$\mathcal{W}2$ runs in six steps. $Step0$'s discretization needs two passes of c cases (one to find mins and maxs, one to convert numerics into B bins). Hence, $Step0$ takes times $2c$.

$Step1$ also takes time $2c$ to generate randomized train and test sets: one pass to randomize the order of the cases, and another to split into train and test. Assuming the use of Fisher-Yates shuffle [52][1], randomization takes time linear on c.

$Step2$ and $Step5$ requires a third pass of the train/test set to collect the K_1 neighbors. These passes do not need to sort all cases- just those in a "best-of" buffer of size K_1. After the $Step1$ randomization, quicksort can sort the neighborhood in time, on average, $2K_1 log(K_1)$ (one for train, one for test).

$Step3$ divides the neighborhood into $best$ and $rest$- a process that takes time K_1. Then $Step4$ takes time equal to:

- the number of ranges to be sorted; i.e. the number of project descriptors times the number of ranges per descriptor (pB);
- plus the log-linear time required to sort those ranges.
- which equals time $pB + pB \cdot log(pB)$.

$Step5$ requires one pass thought the K_1 members of the train data for each candidate treatment. Given pB ranges, the maximum number of pass is pBK_1.

Finally, $Step6$ repeats the above steps n times to check for stability, That is, $\mathcal{W}2$ takes total time:

$$
\begin{array}{ll}
2c+ & ; Step0 \\
n * (& ; Step6\text{'s repeats} \\
\quad 2c+ & ; Step1 \\
\quad 2K_1 log(K_1)+ & ; Step2 \\
\quad K_1+ & ; Step3 \\
\quad pB + pB \cdot log(pB)+ & ; Step4 \\
\quad pBK_1 & ; Step5 \\
) &
\end{array}
$$

This is not a slow process. Nothing in this expression is worse that log-linear. Also, typical values for the n, c, K_1, B, p constants are than $20, 100, 20, 5, 20$ (respectively); i.e. not overly large. Even when implemented in a slow interpreted

1. for $i = n - 1$ to 1: j=random in $0 \leq j \leq i$; swap $item_j$ with $item_i$

language (GAWK), $\mathcal{W}2$ runs in less than half a second for data sets up to 500 cases[2].

4 MODEL- VS INSTANCE-BASED METHODS

$\mathcal{W}2$ was tested against the SEESAW model-based method as follows. SEESAW can only handle models in the COCOMO format. Hence, for this comparison, we restrict ourselves to data in that format. $\mathcal{W}2$ used the historical cases from the NASA93 and COCOMO81 datasets. These data sets all use the features defined by Boehm [43]; e.g. analyst capability, required software reliability, memory constraints, and use of software tools. For this study, we translated them from Boehm original COCOMO format into COCOMOII in order to add estimated defect and development length data for multiple goal optimizations.

Both SEESAW and $\mathcal{W}2$ guided their search using Equation 7 and the four $context$s of §2.2. SEESAW used those $context$s to guide their "what-if" queries around its COCOMO/COQUALMO models. $\mathcal{W}$ used those $contexts$, with NASA93 and COCOMO81, using the six step procedure described above. Recall that, in those steps, some R_x was assessed on projects similar to the $context$ in a $test$ set; i.e. all the cases in the $context$'s neighborhood.

Our comparison rig studied that same $test$ neighborhood using SEESAW. We say that $rows_1$ are the rows selected from the neighborhood after applying SEESAW's recommendations as an SQL select statement. Also, $rows_2$ are the rows selected by $\mathcal{W}2$ from the neighborhood of the $context$ in the $test$ set using the stable treatment found in $Step6$.

SEESAW and $\mathcal{W}2$ were then assessed according to which method returned the better set of $values$. That is, from $rows_i$, we applied Equation 7 to find $values_i$.

The NASA93 results are shown in Figure 10, divided into the defect, effort, months reductions see in ground, flight, OSP2 and OSP. The median and spread improvements are shown in the middle columns. The left-hand-side bar chart shows the 25-th to 75-th percentile range, with a black dot marking the median point. A *negative* improvement denotes an failure to find an improvement. Note that such negative results occur only in a single result (COCOMO81, OSP2, effort).

The "Win" column of those figures indicates when any member of a pair that was both statistically and significantly different. Note that for most pairs, the results are not statistically significantly different (Mann-Whitney, 95% confidence level). The COCOMO81 comparisons are shown in Figure 11. These are in the same format as Figure 10. Sometimes SEESAW's recommendations were unusually bad for COCOMO81, and in those circumstances, often selected nothing from the test neighborhood. This can we seen with $median = 0$ improvement results seen with OSP (defects) and OSP2 (defect and effort).

Before commenting on SEESAW vs $\mathcal{W}2$, we first note that our results should encourage more use of automatic tools for finding changes to software projects. Observe that, in the majority of cases, both model-based and instance-based methods found ways to decrease *both* the median and spread

2. On a 3MHz dual core Macintosh running OS/X 10.6 with 4GB of ram.

Win	Goal	Treatment	Median Reduc	Spread Reduc	Reduction Quartiles 50%
Nasa93 Ground					
	defects	SEESAW	58%	76%	
	defects	W2	21%	53%	
	effort	SEESAW	67%	71%	
	effort	W2	46%	35%	
	months	SEESAW	32%	33%	
	months	W2	18%	24%	
Nasa93 Flight					
	defects	SEESAW	67%	50%	
	defects	W2	55%	35%	
	effort	SEESAW	70%	44%	
	effort	W2	66%	26%	
	months	SEESAW	35%	21%	
	months	W2	30%	24%	
Nasa93 OSP					
*	defects	W2	53%	46%	
	defects	SEESAW	-13%	35%	
*	effort	W2	54%	56%	
	effort	SEESAW	-2%	68%	
*	months	W2	35%	23%	
	months	SEESAW	-3%	19%	
Nasa93 OSP2					
*	defects	W2	62%	29%	
	defects	SEESAW	53%	32%	
*	effort	W2	61%	35%	
	effort	SEESAW	41%	39%	
	months	W2	35%	21%	
	months	SEESAW	29%	16%	

Fig. 10. NASA93: changes in median and spread.

Win	Goal	Treatment	Median Reduc	Spread Reduc	Reduction Quartiles 50%
Coc81 Flight					
	defects	W2	54%	70%	
	defects	SEESAW	31%	115%	
	effort	W2	51%	76%	
	effort	SEESAW	34%	114%	
*	months	W2	33%	36%	
	months	SEESAW	9%	44%	
Coc81 Ground					
*	defects	W2	59%	65%	
	defects	SEESAW	21%	52%	
*	effort	W2	65%	72%	
	effort	SEESAW	50%	67%	
*	months	W2	41%	38%	
	months	SEESAW	14%	31%	
Coc81 OSP					
*	defects	W2	62%	71%	
	defects	SEESAW	2%	43%	
*	effort	W2	59%	95%	
	effort	SEESAW	22%	41%	
*	months	W2	32%	69%	
	months	SEESAW	4%	17%	
Coc81 OSP2					
	defects	W2	15%	81%	
	defects	SEESAW	3%	88%	
*	effort	W2	17%	115%	
	effort	SEESAW	-51%	0%	
	months	SEESAW	22%	43%	
	months	W2	4%	30%	

Fig. 11. COCOMO81: changes in median and spread.

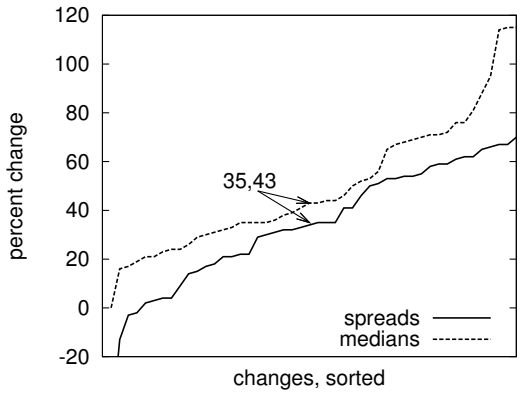


Fig. 12. Range of changes in median and spread generated by applying the recommendations of either $\mathcal{W}2$ or SEESAW. The median observed changes were (34, 34)% for (medians, spreads), respectively. The -51% COC81 OSP2 outlier has been omitted for clarity.

Algorithm	Wins	Losses	Ties
W	13	0	11
SEESAW	0	13	11

Fig. 13. Win/Loss/Tie table from Figure 10 and Figure 11.

of effort/months/defects. In all but one experiment the amount of uncertainty in the median estimates was reduced. As shown in Figure 12, the reduction in the spread was usually over 34%. This is an advantage of these tools since uncertainty is an serious issue that plagues the managers of software engineering projects. As shown in Figure 12, the expected median reduction in any quality estimate was only 34%. Note that if this were otherwise, then that would be a somewhat damning critique of current software engineering practices. To see this, consider the implications of our tools finding recommendations that resulted in an order of magnitude reduction in effort *and* defects *and* development time. That would suggest that the managers of software engineering projects are routinely missing changes that would significantly improve their projects.

In all, we show 24 results:

$$\left(\begin{array}{c} NASA93 \\ COCOMO81 \end{array}\right) * \left(\begin{array}{c} defects \\ effort \\ months \end{array}\right) * \left(\begin{array}{c} ground \\ flight \\ OSP \\ OSP2 \end{array}\right)$$

We conducted statistical tests on each pair of $\mathcal{W}2$ vs SEESAW improvements in median/spread for each query. A Mann Whitney U test (95% confidence) was performed on the two sets of reduction distributions from each comparison. The statistical tests are summarized in Figure 13. Note that, in the majority case ($\frac{18}{24}$), $\mathcal{W}2$'s case-based reasoning performs as well as $SEESAW's$ parametric modeling.

Also, where results differed, instance-based methods can perform much better than model-based. Observe the median reductions for NASA93 OSP: $\mathcal{W}2$ offered median reductions of 72%, 69%, and 43% compared to SEESAW's 22%, 37%, 13% for defects, effort, and months respectively.

In summary, *a simple instance-based methods performs as well, or better, than our best model-based method.*

5 VALIDITY

Construct validity (i.e. face validity) assures that we are measuring what we actually intended to measure [53]. Previous studies have concerned themselves with the construct validity of different performance measures for effort estimation (e.g. [54]). While, in theory, these performance measures have an impact on the rankings of effort estimation algorithms, we have found that other factors dominate. In particular, Figure **??** showed that features of the data set (whether or not it is "weak") have a major impact on what could be concluded after studying a particular estimator on a particular data set. We also show empirically the surprising result that our results are stable across a range of performance criteria.

As to the external validity of our choice of algorithms, recalling Figure **??**, it is clear that this study has not explored the full range of effort estimation algorithms. Clearly, future work is required to repeat this study using the "best of breed" found here (e.g. bands one and two of Figure **??** as well as other algorithms).

Having cast doubts on our selection of algorithms, we hasten to add that this paper has focused on algorithms that have been extensively studied in the literature [55] as well as the commonly available datasets (that is, the ones available in the PROMISE repository of reusable SE data). That is, we assert that these results should apply to much to current published literature on effort estimation.

As to the external validity of our choice of algorithms, recalling Figure **??**, it is clear that this study has not explored the full range of effort estimation algorithms. Clearly, future work is required to repeat this study using the "best of breed" found here (e.g. bands one and two of Figure **??** as well as other algorithms).

Having cast doubts on our selection of algorithms, we hasten to add that this paper has focused on algorithms that have been extensively studied in the literature [55] as well as the commonly available datasets (that is, the ones available in the PROMISE repository of reusable SE data). That is, we assert that these results should apply to much to current published literature on effort estimation.

An unexpected cause for concern with $\mathcal{W}$ was some instability in its recommendations. After removing the nearest neighbor filtering, the *overlap* method for case relevancy filtering helps reduce this affect, as seen in Figure 14. Across four different goals (reducing defects only, effort only, months only, or all at once) $\mathcal{W}2$ returns more stable improvements, whereas $\mathcal{W}$ tends to shift it's recommendations when changing goals.

The above comparison was restricted to data sets with the COCOMO ontology. This section shows that $\mathcal{W}2$ can be applied to much wider range of data sets than the model-based methods.

As shown in Figure 1, we have access to data in a variety of formats:

- Enhancements to a U.K. telecommunications product.
- Projects collected by Miyazaki et al [56] originally comparing "robust regression" to the least squares criteria.
- Finnish Information Systems projects collected by the TIEKE organization.

Stability Comparision for NASA93 FLIGHT																	
goal	*acap=3*	*apex=3*	*apex=5*	*cplx=4*	*data=2*	*data=3*	*ltex=3*	*ltex=4*	*plex=3*	*pmat=2*	*pmat=3*	*pmat=4*	*rely=4*	*rely=4*	*sced=2*	*sced=3*	*stor=3*	*time=3*	*changes*
defects										90						65	75		3
effort										70							90		2
months										70						60	85		3
all										75						70	85		3
Stability Comparision for NASA93 GROUND																			
goal	*acap=3*	*apex=3*	*apex=5*	*cplx=4*	*data=2*	*data=3*	*ltex=3*	*ltex=4*	*plex=3*	*pmat=2*	*pmat=3*	*pmat=4*	*rely=4*	*rely=4*	*sced=2*	*sced=3*	*stor=3*	*time=3*	*changes*
defects										80							50		2
effort										60								75	2
months										75		50						75	3
all										85								80	2
Stability Comparision for NASA93 OSP																			
goal	*acap=3*	*apex=3*	*apex=5*	*cplx=4*	*data=2*	*data=3*	*ltex=3*	*ltex=4*	*plex=3*	*pmat=2*	*pmat=3*	*pmat=4*	*rely=4*	*rely=4*	*sced=2*	*sced=3*	*stor=3*	*time=3*	*changes*
defects	95		70												50				3
effort	80	70													70				3
months												55			80				2
all	50														85		60		3
Stability Comparision for NASA93 OSP2																			
goal	*acap=3*	*apex=3*	*apex=5*	*cplx=4*	*data=2*	*data=3*	*ltex=3*	*ltex=4*	*plex=3*	*pmat=2*	*pmat=3*	*pmat=4*	*rely=4*	*rely=4*	*sced=2*	*sced=3*	*stor=3*	*time=3*	*changes*
defects								60			75				95				3
effort								75			55				80				3
months								70			80				90				3
all								80			90				100				3

Fig. 14. Recommendation frequency across 20 runs of $\mathcal{W}$ and $\mathcal{W}2$ for reducing individual goals (defects, effort, months) as well as all goals at once (d/e/m). Recommendations that appeared less than half the time are omitted. Note that despite chaning goal functions, $\mathcal{W}2$ continues to make similar recommendations.

- A large dataset of Chinese software projects.
- Large COBOL projects, collected by Kemerer [57].

The data sets use a variety of features such as number of entities in the data model, number of basic logical transactions, query count and number of distinct business units serviced.

Given that we did not have access to specific case studies like Figure 4, synthetic queries were developed. Three queries were generated for each of the five datasets. The first contained the entire range of possible project project descriptors, representing complete freedom to recommend any change within the space. The other two queries were generated by randomly choosing 50% of each attribute values from either the lower, middle, or upper ranges for each project descriptor.

The only quality measures available for this data was development effort. Hence, the following results are expressed only in terms of effort.

Effort reductions can be seen in Figure 15:

- Overall, we say a median 56% and 73% improvement in median and spread, respectively (see the dotted line of Figure 15.
- Large improvement in median effort were seen for the TELECOM, KEMERER, and MIYAZAKI datasets, with strong median reductions (>50%) and good spread reductions (>25%) across all datasets.
- The FINNISH and CHINA datasets show good improvements to both median and spread reductions, with some projects demonstrating exceptional reductions in spread (>75%).

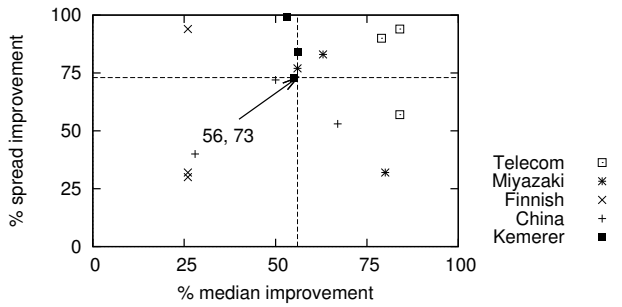


Fig. 15. Distribution of median and spread reductions in software effort for five non-COCOMO datasets.

Note that it would be impossible to apply our model-based methods to these data sets, since they are not expressed in terms of the COCOMO ontology. That is, at least in our experience, *instance-based methods can be applied to more data sets that effort-based methods*.

6 Related Work

This paper extends prior publications on $\mathcal{W}2$ as follows.

Instance-based analysis of software estimation is a widely explored area [36], [50], [51], [55]. We show here that a simple addition to standard instance-based analysis can not just estimates, but also recommendations on how to *change* those estimated. This is an important extension since, in our experience, if a manager is presented with an estimate, their very next question is "what can I do to change this?". Hence, while the effort estimation literature describes many estimation methods, both model-based and instance-based [28], [36], [39], [49]–[51], [55], [58], we focus more on how to *change* those estimates.

With the model-based methods, our challenge was to handle the uncertainty associated with the tuning instability discussed in §2. Previously, we have tried reducing that instability in various ways:

- Feature selection to prune spurious details [59];
- Instance selection to prune irrelevancies [60];
- Extended data collection.

Despite all that work, the variance observed in our models remains very large. Even the application of techniques such as instance-based learning have failed to reduce variance in our effort predictions [61]. Feature subset selection has also been disappointing: while it reduces the median performance variance somewhat (in our experiments, from 150% to 53% [60]), the residual error rates are large enough that it is hard to use the predictions of these models as evidence for the value of some proposed approach. Lastly, further data collection has not proven useful. Certainly, there is an increase in the availability of historical data on prior projects[3]. However, Kitchenham [62] cautions that the literature is contradictory regarding the value of using data from other companies to learn local models.

3. see http://promisedata.org.data/data and http://www.isbsg.org/.

Having failed to tame tuning variance, despite years of research, we turned to alternate methods. Much of the related work on uncertainty in software engineering uses a Bayesian analysis. For example, Pendharkar et al. [63] demonstrate the utility of Bayes networks in effort estimation while Fenton and Neil explore Bayes nets and defect prediction [64] (but unlike this paper, neither of these teams links defect models to effort models).

Other related work is the search-based SE approach advocated by Harman [65]. 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. Harman takes care to distinguish AI search-based methods from those seen in standard numeric optimizations. Such optimizers usually offer settings to all controllables. This may result in needlessly complex recommendations since a repeated empirical observation is that many model inputs are noisy or correlated in similar ways to model outputs [66]. Such noisy or correlated variables can be pruned away to generate simpler solutions that are easier and quicker to understand. In continuous domains, there is much work on feature selection [67] and techniques like principal component analysis [68] to reduce the number of dimensions reported by an analysis. Comparative studies report that discrete AI-based methods can do better at reducing the size of the reported theory [66].

The SBSE approach can and has been applied to many problems in software engineering (e.g., requirements engineering [69]) but most often in the field of software testing [2]. Harman's writing inspired us to try simulated annealing for our model-based methods [19] (which we subsequently found worked worse than SEESAW or $\mathcal{W}2$).

7 Future work

better optimization search

- *Pareto domination* [70] that rejects point X if some other point Y is (a) not worse on any quality dimension and (b) better on at least one.
- Or an *objective function* that combines the individual qualities into a single number; e.g. Equation 6.

contrast set learning, not with this simple Bayesian trick

shown we are better than other some, not better than all

8 Conclusion

if no data, then model-based. but bear in mind all the drawbacks discussed above with that approach.

not "best"- only better than anything we've seen before.

Exploring software project *process* data may not be as complex as exploring software project *product* data. For example, two developers, working for one year on one project, can generate defect data on hundreds to thousands of software *products* (e.g. each module they have built). However, that same work might contribute only one entry to a software *process* data base (e.g. one row describing their applications experience with this kind of software, the size of the final system, and the number of staff months required to code it all). For example:

- The effort estimation datasets used in Mendes et al. [36], Auer et al. [37], Baker [38], and Li et al. [39] have median number of examples 13,15,33,52 (respectively).
- the experiments of this paper were conducted on seven projects containing 15,18,38,48,62,93,499 examples (i.e. usually less than a few dozen examples per project).

For complex software *product* data, it may be advisable to apply support vector machines, random forests, or any of the other sophisticated analysis methods surveyed in, say, [29]. However, for software *process* data, it may be overkill to apply complex tools like the various Davis-Putnam procedures; binary-decision diagrams; integer programming; genetic programming; tabu search; or the hundreds of other tools described in the AI literature [12], [30]–[33] or the search-based software engineering literature [33].

This is not to say that learning changes to software projects is *always* best achieved using simple instance methods. For example, the next release planning problem studied by Ruhe [35] (and others) is a process problem of great complexity. Hence, the Pareto frontier optimization methods employed by Ruhe may be an appropriate technology for that domain.

Nevertheless, other areas of research advise that researchers should at baseline their supposedly more sophisticated method against a simpler alternative [25], [26]. Accordingly, we offer $\mathcal{W}2$ as a baseline device for learning changes to software projects since:

- It is simple to implement and fast to execute,
- The algorithm runs in linear time, so it can scale to large data sets.
- For the data studied here, it performs as least as good (or better) than a range of model-based methods.

References

[1] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, and M.-Y. Wong, "Orthogonal defect classification–a concept for in-process measurements," *IEEE Transactions on Software Engineering*, vol. 18, no. 11, pp. 943–956, November 1992.

[2] 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.

[3] G. Holzmann, "The model checker SPIN," *IEEE Transactions on Software Engineering*, vol. 23, no. 5, pp. 279–295, May 1997.

[4] D. Port, A. Olkov, and T. Menzies, "Using simulation to investigate requirements prioritization strategies," in *IEEE ASE'08*, 2008, available from http://menzies.us/pdf/08simrequire.pdf.

[5] R. A. Endres, H.D, *A Handbook of Software and Systems Engineering: Empirical Observations, Laws and Theories*. Addison Wesley, 2003.

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

[7] 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.

[8] B. Boehm, "Software engineering," *IEEE Transactions on Computers*, vol. 25, pp. 1226–1241, 1976.

[9] T. Menzies, S. Williams, O. El-rawas, B. Boehm, and J. Hihn, "How to avoid drastic software process change (using stochastic statbility)," in *ICSE'09*, 2009, available from http://menzies.us/pdf/08drastic.pdf.

[10] N. E. Fenton, M. Neil, and J. G. Caballero, "Using ranked nodes to model qualitative judgments in bayesian networks," *IEEE Trans. on Knowl. and Data Eng.*, vol. 19, no. 10, pp. 1420–1432, 2007.

[11] I. H. Witten and E. Frank, *Data mining. 2nd edition*. Los Altos, US: Morgan Kaufmann, 2005.

[12] S. J. Russell, P. Norvig, J. F. Candy, J. M. Malik, and D. D. Edwards, *Artificial intelligence: a modern approach*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2003.

[13] M. J. Shepperd, "Case-based reasoning and software engineering," Bournemouth University, UK, Tech. Rep. TR02-08, 2002.

[14] D. B. Leake, *Case-Based Reasoning: Experiences, Lessons and Future Directions*. Cambridge, MA, USA: MIT Press, 1996.

[15] J. Kolodner, *Case-Based Reasoning*. Morgan Kaufmann, 1993.

[16] D. Leake and D. Mcsherry, "Introduction to the special issue on explanation in case-based reasoning," *Artificial Intelligence Review*, vol. 24, pp. 103–108, 2005.

[17] T. Menzies, "The complexity of trmcs-like spiral specification," in *Proceedings of 10th International Workshop on Software Specification and Design (IWSSD-10)*, 2000, available from http://menzies.us/pdf/00iwssd.pdf.

[18] 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.

[19] 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*. New York, NY, USA: ACM, 2007, pp. 303–312, available from http://menzies.us/pdf/07casease-v0.pdf.

[20] A. Orrego, T. Menzies, and O. El-Rawas, "On the relative merits of software reuse," in *International Conference on Software Process*, 2009, available from http://menzies.us/pdf/09reuse.pdf.

[21] T. Menzies, S. Williams, O. Elrawas, D. Baker, B. Boehm, J. Hihn, K. Lum, and R. Madachy, "Accurate estimates without local data?" *Software Process Improvement and Practice*, vol. 14, pp. 213–225, July 2009, available from http://menzies.us/pdf/09nodata.pdf.

[22] T. Menzies, O. El-Rawas, J. Hihn, and B. Boehm, "Can we build software faster and better and cheaper?" in *PROMISE'09*, 2009, available from http://menzies.us/pdf/09bfc.pdf.

[23] P. Green, T. Menzies, S. Williams, and O. El-waras, "Understanding the value of software engineering technologies," in *IEEE ASE'09*, 2009, available from http://menzies.us/pdf/09value.pdf.

[24] O. El-Rawas and T. Menzies, "A second look at faster, better, cheaper," *Innovations in Systems and Software Engineering*, 2011.

[25] P. Cohen, *Empirical Methods for Artificial Intelligence*. MIT Press, 1995.

[26] R. Holte, "Very simple classification rules perform well on most commonly used datasets," *Machine Learning*, vol. 11, p. 63, 1993.

[27] P. Domingos and M. J. Pazzani, "On the optimality of the simple bayesian classifier under zero-one loss," *Machine Learning*, vol. 29, no. 2-3, pp. 103–130, 1997. [Online]. Available: citeseer.ist.psu.edu/domingos97optimality.html

[28] M. Jorgensen and M. Shepperd, "A systematic review of software development cost estimation studies," January 2007, available from http://www.simula.no/departments/engineering/publications/Jorgensen.2005.12.

[29] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Transactions on Software Engineering*, 2008.

[30] J. Gu, P. W. Purdom, J. Franco, and B. W. Wah, "Algorithms for the satisfiability (sat) problem: A survey," in *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1997, pp. 19–152.

[31] G. Gay, T. Menzies, O. Jalali, G. Mundy, B. Gilkerson, M. Feather, and J. Kiper, "Finding robust solutions in requirements models," *Automated Software Engineering*, vol. 17, no. 1, pp. 87–116, 2010, available from http://menzies.us/pdf/09keys2.pdf.

[32] T. E. Uribe and M. E. Stickel, "Ordered binary decision diagrams and the davis-putnam procedure," in *In Proc. of the 1st International Conference on Constraints in Computational Logics*. Springer-Verlag, 1994, pp. 34–49.

[33] M. Harman, "The current state and future of search based software engineering," in *Future of Software Engineering, ICSE'07*, 2007.

[34] Y. Zhang, M. Harman, and S. Mansouri, "The multi-objective next release problem," in *In ACM Genetic and Evolutionary Computation Conference (GECCO 2007*, 2007, p. 11.

[35] A. Ngo-The and G. Ruhe, "Optimized resource allocation for software release planning," *Software Engineering, IEEE Transactions on*, vol. 35, no. 1, pp. 109–123, Jan.-Feb. 2009.

[36] E. Mendes, I. D. Watson, C. Triggs, N. Mosley, and S. Counsell, "A comparative study of cost estimation models for web hypermedia applications," *Empirical Software Engineering*, vol. 8, no. 2, pp. 163–196, 2003.

[37] M. Auer, A. Trendowicz, B. Graser, E. Haunschmid, and S. Biffl, "Optimal project feature weights in analogy-based cost estimation: Improvement and limitations," *IEEE Trans. Softw. Eng.*, vol. 32, pp. 83–92, 2006.

[38] 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.

[39] Y. Li, M. Xie, and T. Goh, "A study of project selection and feature weighting for analogy based software cost estimation," *Journal of Systems and Software*, vol. 82, pp. 241–252, 2009.

[40] A. Brady and T. Menzies, "Case-based reasoning for reducing software development effort," *Journal of Software Engineering Applications*, December 2010, available from http://menzies.us/pdf/10w0.pdf.

[41] ——, "Case-based reasoning vs parametric models for software quality optimization," in *PROMISE '10: Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, 2010, pp. 1–10, available from http://menzies.us/pdf/10cbr.pdf.

[42] 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.

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

[44] ——, "Safe and simple software cost analysis," *IEEE Software*, pp. 14–17, September/October 2000, available from http://www.computer.org/certification/beta/Boehm_Safe.pdf.

[45] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science, Number 4598, 13 May 1983*, vol. 220, 4598, pp. 671–680, 1983. [Online]. Available: citeseer.nj.nec.com/kirkpatrick83optimization.html

[46] B. Selman, H. A. Kautz, and B. Cohen, "Local search strategies for satisfiability testing," in *Proceedings of the Second DIMACS Challange on Cliques, Coloring, and Satisfiability*, M. Trick and D. S. Johnson, Eds., Providence RI, 1993. [Online]. Available: citeseer.ist.psu.edu/selman95local.html

[47] S. Craw, D. Sleeman, R. Boswell, and L. Carbonara, "Is knowledge refinement different from theory revision?" in *Proceedings of the MLNet Familiarization Workshop on Theory Revision and Restructuring in Machine Learning (ECML-94)*, S. Wrobel, Ed., 1994, pp. 32–34.

[48] W. Cohen, "Fast effective rule induction," in *ICML'95*, 1995, pp. 115–123, available on-line from http://www.cs.cmu.edu/~wcohen/postscript/ml-95-ripper.ps.

[49] U. Lipowezky, "Selection of the optimal prototype subset for 1-NN classification," *Pattern Recognition Letters*, vol. 19, p. 907918, 1998. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0167865598000750

[50] F. Walkerden and R. Jeffery, "An empirical study of analogy-based software effort estimation," *Empirical Softw. Engg.*, vol. 4, no. 2, pp. 135–158, 1999.

[51] C. Kirsopp and M. Shepperd, "Making inferences with small numbers of training sets," *IEEE Proc.*, vol. 149, 2002.

[52] R. Durstenfeld, "Algorithm 235: Random permutation," *Commun. ACM*, vol. 7, no. 7, p. 420, 1964.

[53] C. Robson, "Real world research: a resource for social scientists and practitioner-researchers," *Blackwell Publisher Ltd*, 2002.

[54] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit, "A simulation study of the model evaluation criterion mmre," *IEEE Transactions on Software Engineering*, vol. 29, no. 11, pp. 985 – 995, November 2003.

[55] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," *IEEE Transactions on Software Engineering*, vol. 23, no. 12, November 1997, available from http://www.utdallas.edu/~rbanker/SE_XII.pdf.

[56] Y. Miyazaki, M. Terakado, K. Ozaki, and H. Nozaki, "Robust regression for developing software estimation models," *J. Syst. Softw.*, vol. 27, no. 1, pp. 3–16, 1994.

[57] C. Kemerer, "An empirical validation of software cost estimation models," *Communications of the ACM*, vol. 30, no. 5, pp. 416–429, May 1987.

[58] M. Shepperd, "Software project economics: A roadmap," in *International Conference on Software Engineering 2007: Future of Software Engineering*, 2007.

[59] 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.

[60] 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.
[61] O. Jalali, "Evaluation bias in effort estimation," Master's thesis, Lane Department of Computer Science and Electrical Engineering, West Virginia University, 2007.
[62] B. A. Kitchenham, E. Mendes, and G. H. Travassos, "Cross- vs. within-company cost estimation studies: A systematic review," *IEEE Transactions on Software Engineering*, pp. 316–329, May 2007.
[63] P. C. Pendharkar, G. H. Subramanian, and J. A. Rodger, "A probabilistic model for predicting software development effort," *IEEE Trans. Softw. Eng.*, vol. 31, no. 7, pp. 615–624, 2005.
[64] N. E. Fenton and M. Neil, "A critique of software defect prediction models," *IEEE Transactions on Software Engineering*, vol. 25, no. 5, pp. 675–689, 1999, available from http://citeseer.nj.nec.com/fenton99critique.html.
[65] 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*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 728–729.
[66] M. Hall and G. Holmes, "Benchmarking attribute selection techniques for discrete class data mining," *IEEE Transactions On Knowledge And Data Engineering*, vol. 15, no. 6, pp. 1437– 1447, 2003, available from http://www.cs.waikato.ac.nz/~mhall/HallHolmesTKDE.pdf.
[67] A. Miller, *Subset Selection in Regression (second edition)*. Chapman & Hall, 2002.
[68] W. Dillon and M. Goldstein, *Multivariate Analysis: Methods and Applications*. Wiley-Interscience, 1984.
[69] 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.
[70] J. Li and G. Ruhe, "Multi-criteria decision analysis for customization of estimation by analogy method aqua+," in *PROMISE '08: Proceedings of the 4th international workshop on Predictor models in software engineering*, 2008, pp. 55–62.

Tim Menzies is a associate professor at the Lane Department of Computer Science at West Virginia University (USA), and has been working with NASA on software quality issues since 1998. He has a CS degree and a PhD from the University of New South Wales and is the author of over 160 publications. His recent research concerns modeling and learning with a particular focus on light weight modeling methods.

Adam Brady is a associate professor at the Lane Department of Computer Science at West Virginia University (USA), and has been working with NASA on software quality issues since 1998. He has a CS degree and a PhD from the University of New South Wales and is the author of over 160 publications. His recent research concerns modeling and learning with a particular focus on light weight modeling methods.

Jacky W. Keung is a associate professor at the Lane Department of Computer Science at West Virginia University (USA), and has been working with NASA on software quality issues since 1998. He has a CS degree and a PhD from the University of New South Wales and is the author of over 160 publications. His recent research concerns modeling and learning with a particular focus on light weight modeling methods.

Steve Williams is a associate professor at the Lane Department of Computer Science at West Virginia University (USA), and has been working with NASA on software quality issues since 1998. He has a CS degree and a PhD from the University of New South Wales and is the author of over 160 publications. His recent research concerns modeling and learning with a particular focus on light weight modeling methods.

Steven Williams is an undergraduate student pursuing a BS in Computer Science at Portland State University. He is currently working as an IT Manager and Database Administrator at a large non-profit organization in Portland, Oregon.

Oussama El-Rawas is a associate professor at the Lane Department of Computer Science at West Virginia University (USA), and has been working with NASA on software quality issues since 1998. He has a CS degree and a PhD from the University of New South Wales and is the author of over 160 publications. His recent research concerns modeling and learning with a particular focus on light weight modeling methods.

Phillip Green is a graduate student in the Computer Science Department at Portland State University. He received his BS in Physics and Astronomy from the University of Pittsburgh in 2001. He has over six years of research experience in numeric methods and data mining. His master thesis focuses on comparative study of data mining techniques and equivalences with numeric optimization techniques. He also has interned at a software development firm in Beaverton, Oregon.