Learning Changes to Software Projects

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

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 any other methods 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 and easy to maintain (just add more instances).
CONCLUSION: We recommend instance-based methods such as $\mathcal{W}2$ for learning changes to a software project.

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

1 INTRODUCTION

There are many ways a manager might *change*, and hopefully *improve*, their software development project. Some changes require *tools* such as using the new generation of functional programming languages or execution and testing tools [1] or automated formal analysis [2]. Other changes use *process improvement* techniques such as changing the organizational hiring practices, or a continual renegotiation of the requirements as part of an agile software development cycle [3].

Endres & Rombach [4] list dozens of *laws* of software engineering to justify a particular change to a project. If a manager proposed using *all* the laws, then senior management would most likely suggest they scale back their plans to just *a minimal set of most effective measures*.

This paper explores different ways for finding this minimal set of most effective changes to a project. Specifically, we compare *model-based* vs *instance-based* methods. The difference between these two methods is as follows. Model-based methods develop a model via expert advice [5] or using automatic methods such as *data mining* [6]. Once built, the model can be used for "what-if" queries in order to assess possible changes to a project. For example:

$$\begin{array}{rcl} data & \rightarrow & model \\ model + whatIf & \rightarrow & scores \end{array}$$

Here, $Scores$ represents business concerns; for example reduce defects before release the software product. Also, the $whatIf$ query defines a $context$ within which a manager seeks ways to improve a project.

Instance-based methods, on the other hand, insert the "what-if" query into a n-dimensional space populated with historical project cases [7]–[11]. Unlike model-based methods, instance-based methods do not require an underlying model. Rather, the immediate neighborhood of the "what-if" is somehow scored to find summary of those neighboring cases:

$$\begin{array}{rcl} data + whatIf & \rightarrow & neighborhood \\ neighborhood & \rightarrow & scores \end{array}$$

In previous work [12]–[20]. we tried combining model-based methods with AI tools to control thousands of "What-If" queries over COCOMO models. This paper compares those model-based methods with "$\mathcal{W}2$", a novel instance-based method. Given a "What-If" query that selects some set of similar projects, $\mathcal{W}2$ seeks a $treatment\ R_x$, which finds the "better" parts of those similar projects within the dataset:

$$\begin{array}{rcl} data + WhatIf & \rightarrow & neighborhood_1 \\ neighborhood_1 & \rightarrow & scores_1 \\ R_x + data + WhatIf & \rightarrow & neighborhood_2 \\ neighborhood_2 & \rightarrow & scores_2 \\ scores_2 & > & scores_1 \end{array}$$

When compared to model-based methods:

- $\mathcal{W}2$ identified *similar* or *better* treatments.
- $\mathcal{W}2$ *was faster to run:* all the experiments in this paper require 10 minutes with $\mathcal{W}2$, but days for using models.

- *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 Department of Computing, The Hong Kong Polytechnic University. E-mail: Jacky.Keung@comp.polyu.edu.hk.*
- *Steven Williams is with the School of Informatics and Computing Indiana University, Bloomington. E-mail: stevencwilliams@gmail.com.*
- *Jairus Hihn is with Caltech's Jet Propulsion Laboratory. E-mail: jairus.hihn@jpl.nasa.gov.*
- *Barry Boehm is with the University of Southern California. E-mail: boehm@sunset.usc.edu.*

This research was conducted at West Virginia University, University of Southern California, and NASA Jet Propulsion Laboratory under a NASA sub-contract. Reference herein to any specic commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government.
This research was funded in part by NSF,CISE, project #0810879.

- $\mathcal{W}2$ *was simpler to implement:* $\mathcal{W}2$'s 200 lines AWK replaces thousands of lines of the model-based LISP.
- $\mathcal{W}2$ *was simpler to maintain:* with instance-based methods, "maintenance" implies "adding more instances".
- $\mathcal{W}2$ *was simpler to adapt to new domains:* $\mathcal{W}2$ do not require an underlying model and therefore it imposes no restrictions on the data being processed. It is more efficient and it can be quickly applied to more data sets.

Hence, we now recommend the instance-based methods like $\mathcal{W}2$ for identifying changes to software projects.

In summary, this study has 4 significant contributions:

1) We demonstrate that managers have many options for effectively changing and improving their projects.
2) We enhance algorithms proposed by other researchers. Prior work on instance-based methods founds ways to $generate$ estimates [21]–[24]. We show that a small modification to standard instance-based analysis allows us to determine how to $change$ an estimate.
3) We improve our prior results on instance-based methods. $\mathcal{W}2$ outperforms our model-based methods described in [12]–[20] as well as earlier versions of our instance-based reasoner in [25] and [26]. As discussed in §3, $\mathcal{W}2$ handles missing values better than $\mathcal{W}1$. More significantly, this paper is the first report on extending any version of $\mathcal{W}$ beyond standard COCOMO data.
4) Results show simple instance-based methods can perform better than more complex model-based methods.

While the first points may be of most interest to industrial practitioners, but it is the last point that may be most interesting to researchers. There are many sophisticated methods for exploring the complexities and uncertainties of trying to control software engineering projects. Based on the results of this paper, we advise researchers to first explore simpler methods, if only for the purposes of establishing a performance baseline.

This result shows that the $\mathcal{W}2$ instance-based method is superior to all the model-based methods explored in this study. This does not imply that learning changes to software projects is *always* best achieved using simple instance methods. For example, the next release planning problem discussed in [27], [28] is a process problem of great complexity. For that task, the Pareto frontier optimization methods employed by (e.g.) Ruhe [27] is preferred to $\mathcal{W}2$.

2 BACKGROUND

2.1 Software Estimation Research

Instance-based software estimation such as *Case-Based Reasoning* (CBR) is a widely explored area in software engineering research [21]–[24]. Based on our collective experience, when a manager sees an estimate, his/her immediate question is "how can I change that?". While the effort estimation literature describes many estimation methods (both model-based and instance-based [21]–[24], [29]–[33]) in order to address manager's immediate concern, we focus on how to *change* estimates.

$\mathcal{W}2$ explores multiple goals such as reducing development effort *and* defects *and* the total calendar time to deliver the software. Instead, most other work in this area explores a single goal. For example, Pendharkar et al. [29] demonstrate the utility of Bayes networks in software effort estimation while Fenton and Neil explore Bayes nets and software defect prediction [34], neither of these teams links defect models to effort models. In addition, as mentioned above, these work focus much more on *prediction*, rather than on the subsequent problem of learning how to *change* those predictions.

2.2 Search-Based Software Engineering (SBSE)

Multi-goal optimization in Search-Based Software Engineering (SBSE) is well explored in the field [35]. SBSE employs optimization techniques from operations research and meta-heuristic search (for example in simulated annealing and genetic algorithms) in an attempt to hunt for near-optimal solutions. Harman [35] distinguishes AI search-based methods from those seen in standard numeric optimizations. Such optimizers offer settings to all controllables. This may result in needlessly complex recommendations since a repeated empirical observation is that many model inputs are contaminated or correlated in similar ways to model outputs [36]. Such contaminated or correlated variables can be pruned to generate simpler solutions that are easier and quicker to understand. For continuous variables, there are many work on feature selection [37] and techniques like principal component analysis [38] to reduce the number of dimensions reported by a data analysis. Some studies report that discrete AI methods perform better at reducing the size of the reported theory [36].

The SBSE approach can and has been applied successfully to many software engineering domains such as requirements engineering [39], but more commonly used in software testing [1]. Harman's work provides the inspiration to this study in an attempt to experiment simulated annealing for our model-based methods [14] (which we subsequently found performed worse than $\mathcal{W}2$).

2.3 Model: Benefits

High-level abstraction models represent and transmit common software patterns observed in multiple specific situations [40]. At a keynote address at PROSIM'05 Walt Scacchi noted that merely writing a model can clarify local business processes [41]. Executable software process models can be used for many purposes including but not limited to reducing the inspection effort at different stages of the software life cycle [42]. Even if a model lacks a sophisticated execution engine, it can still be used for what-if queries that are insightful to different business processes (e.g. see Boehm et al.'s what-if studies in Chapter Three of [43]).

Models can combine and summarize *both* expert insights *and* local data. Fenton [5] builds the general structure of his Bayes nets via workshops of business knowledge. The details of these structures are then tuned via local data. Elsewhere, Boehm reports the advantages of combining local data with model structures initialized via expert knowledge [44].

Another subtle advantage of models is data sharing. Schulz reports that organizations that are reluctant to share specific

Dataset	Cols	Rows	Notes	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
COC81ii	26	63	NASA projects	effort, time, defects
NASA93ii	26	93	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.

decreases effort	acap: analyst capability
	apex: applications experience
	ltex: language and tool-set experience
	pcap: programmer capability
	pcon: personnel continuity
	plex: platform experience
	site: multi-side development
	tool: use of software tools
	sced: dictated development schedule
increases effort	cplx: product complexity
	data: database size
	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.

data, may be willing to share models (if those models do not reveal details from particular business sites) [45].

Finally, models let us extrapolate from past examples to new examples. A trend that is sampled by N historical examples can be extended to offer predictions for new examples that have not been seen previously.

2.4 Models: Drawbacks

Extrapolation, while sometimes useful, may over fit the data. If that occurs, then a model may offer unsupported recommendations. For example, as shown in the results below, our model-based methods were ineffective since, sometimes, they proposed conclusions that applied to *none* of the test data.

Another drawback with model-based tools is that they 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. For example, Figure 1 shows the data sets used in this study. Our model-based methods can only process the two data sets that conform to the COCOMO ontology of Figure 2. On the other hand, the $\mathcal{W}2$ instance-based method can process all of them.

Models need to be learned from data and collecting that data can be difficult due to the business sensitivity associated with the data as well as differences in how the metrics are determined, collected and archived. In many cases the required data is not archived at all. In our experience, for example, after two years we were only able to add 7 records to our NASA wide software cost metrics repository [14]. Alternatively, open-source code repositories are a rich source of *product* information, but usually lack *process details* such as the descriptions of the applications experience of the developers. Other researchers also have noted similar problems with collecting process data. Lowry [46] discusses the complexities involved in calibrating his software failure models. Those models require parameters that are clearly antiquated. For example, he mentions a commercial model-based cost estimation tool that requires a parameter that rates "the time it takes for a software development environment to respond to a keyboard input". When software was written on remote time-shared computers, this was an important factor. However, today it is irrelevant but it is kept in the model for backwards compatibility and because it was measured in the software projects on which the model was calibrated.

Baker [47] discusses another serious concerns with model calibration: *tuning instability*. Software construction is a very human-intensive process, therefore the data collected from that process is as varied as the humans building the code. Consider the following simplified COCOMO [43] model:

$$effort = a \cdot LOC^{b+pmat} \cdot acap \quad (1)$$

The equation presents COCOMO's core assumption that software development effort is exponential on software size. In this equation, a and b control the linear and exponential inferences (respectively) on model estimates; while $pmat$ (process maturity) and $acap$ (analyst capability) are project choices articulated by managers. Equation 1 contains only two features ($acap, pmat$) and a full COCOMO model contains a set of project descriptors as shown in Figure 2.

Baker [47] learned values of (a, b) for a full COCOMO model using Boehm's local calibration method [48] from 300 random samples of 90% of the available project data. The ranges varied widely:

$$(3.2 \leq a \leq 9.4) \wedge (0.8 \leq b \leq 1.12) \quad (2)$$

Such large variation in model tunings not only violates standard gradient descent methods, but it also obscure any benefits observed within a particular project change. Suppose a proposed technology doubles productivity, but a changed from 9.0 to 4.5, any improvement would be obscured by the *tuning instability*.

In summary, model-based methods can suffer from:

- Inappropriate extrapolations;
- Ontology restrictions;
- Untamed variance inside the models

Hence, in this paper, we explore alternative methods.

3 Instance-based Methods

This section described three versions of the $\mathcal{W}$ instance-based tool. Lessons learned from $\mathcal{W}0$ [25] and $\mathcal{W}1$ [26] will inform the description of the current version, $\mathcal{W}2$.

Similar to 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 datasets 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: NASA93ii and COC81ii. $\mathcal{W}$'s applicability to a wide

Figure 3.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 3.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 3.c: Rank with Equation 3.

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. 3. Processing $context_1$ on a training set selected from NASA93ii.

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 of these qualities are summarized into a single value by some $value$ function like Equation 6.

Similar to our model-based methods, $\mathcal{W}$ assumes that a manager can offer us (a) a description of the $context \subseteq P$ that interests them and (b) 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 of assigning inexperienced people. The manager commented that, at his 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 $control$labels; 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.

Another important assumption made by $\mathcal{W}2$ is that we should not reason on *all* the data. Rather, we need to focus on the data *relevant* to the $context$ of the current problem. A mistake made by $\mathcal{W}0$ was to reason over all the data- which lead to problems of learning from inappropriate examples. $\mathcal{W}0$ tended to converge on nearby projects with increased productivity, but because it had (e.g.) a lower level of complexity or required reliability, it selected for regions containing acceptable alternatives. Sometimes, this is unavoidable (e.g. if all the available examples mention lower complexity or reliability). However, as much as possible, $\mathcal{W}$'s reasoning needs to respect the $context$ limitations offered by the user.

$\mathcal{W}2$ finds a project treatment R_x by studying the project similar to the context in the case repository. Formally, $\mathcal{W}2$ explores the $neighorhood$ of the $context$, looking for ways to select for the "best" cases. (as determined by a $value$ functions like Equation 6). In $\mathcal{W}2$, this is a seven step procedure:

1) Divide cases randomly into $train : test$ in the ratio 2:1.
2) Use $context$ to find the neighborhood within $train$.
3) Divide neighborhood into (a) the $best$ cases that should be emulated, and (b) the remaining cases to avoid (which we call $rest$).
4) Rank all differences between (a) and (b) according to how strongly they select for the $best$ cases.
5) Use 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) Test the treatment from $Step6$ using relevant cases from the $test$ set; i.e. find all rows in the neighborhood of the $context$ in $test$ set; then find the subset of those rows that match the treatment. Assess those rows using a $value$ function such as Equation 6.
7) Repeat above six steps $n = 20$ times with other randomly selected $train : test$ sets. Prune $unstable$ treatments; i.e. those not found in the majority of repeats.

Note that $Step7$ is a new feature of $\mathcal{W}2$ (since $\mathcal{W}0$ and $\mathcal{W}1$ neglected to test for treatment stability).

On 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 3.a shows training data from NASA93ii. The gray cells in that figure show ranges that do not appear in $context_1$. Note that 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 solves the problem of missing parts of the $context$. However, it creates another problem: a large variance in all the results. To rectify this issue, $\mathcal{W}2$ ensures $context$ became a set overlap membership function. For example, $context_1$ defines ranges for 14 project descriptors. Any training case contains ranges that overlaps with between 0 and 14 of the ranges is $context_1$. As shown in Figure 3.a, we can use the size of this overlap to sort the cases: the neighborhood cases are those with greatest overlap. Appealing to the central limit theorem, $\mathcal{W}2$ implies that the neighborhood of a query are the $K_1 = 20$ training cases with largest overlap to the query (and for very small data sets, we use all the training examples except the $K_2 = 5$ examples discussed below).

$\mathcal{W}2$'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 $round((y - min)/((max - min)/B))$. We set the B value after experiments with one dataset, then leaving it fixed for the rest. For all of the $\mathcal{W}$ experiments in this paper, $B = 5$.

$Step3$ (division into $best$ and $rest$) is illustrated in Figure 3.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 [22]–[24], [30], [31] 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 3.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

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. 4. A $K_1 = 20$ neighborhood of $context_1$ (NASA93ii train set).

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. 5. All rows of Figure 4 satisfying $R_1 : pmat = 3$.

row	apex	plex	ltex	pmat	rely	data	cplx	time	stor	pvol	acap	pcap	tool	sced	effort	overlap
11	5	3	4	3	3	2	4	3	3	2	4	4	3	3	24	10
15	5	3	4	3	3	2	4	3	3	2	4	4	3	3	48	10
19	5	3	4	3	3	2	4	3	3	2	4	4	3	3	48	10
18	4	3	4	3	3	2	4	3	3	2	4	4	3	3	60	10
10	5	3	4	3	3	2	4	3	3	2	4	5	3	3	72	10
24	4	3	3	3	4	3	3	4	3	3	3	4	3	3	90	11
63	4	3	4	3	3	3	3	3	3	2	4	4	3	3	162	9
45	4	3	4	3	4	4	3	3	3	2	3	4	3	3	400	8
67	4	3	4	3	5	3	4	4	3	2	4	4	3	3	444	11
60	3	4	4	3	3	2	4	3	3	2	5	5	3	3	720	10

Fig. 6. The testing set with all cases not containing $pmat = 3$ removed.

ranking formulae is:

$$rank(E) * support(E) = \frac{ratio(E|best)^2}{ratio(E|best) + ratio(E|rest)} \quad (3)$$

The x top-ranked items of Figure 3.c are candidate treatments:

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

$Step5$ (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 4 shows the $K_1 = 20$ cases closest to $context_1$ in the train set.

Figure 5 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 5 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, $Step5$ explores R_x upwards from $x = 1$ while:

- The median $value$ of the rows selected by R_{x+1} is greater than that of R_x.
- The number of selected rows $|R_x \wedge neighborhood| \geq 3$; In the case of Figure 5, $Step5$ returns R_1 ($pmat = 3$).

Figure 6 applies R_1 on the test data. Its impact is reported as the median effort value of the cases. For the cases of Figure 6, this is 81 months of effort.

$\mathcal{W}2$ is not a slow algorithm. Nothing in any of these steps takes more than log-linear time, and even that is only to sort K_1 items (which is a very small list). Even when implemented in an interpreted language (GAWK), $\mathcal{W}2$ runs in less than half a second for up to 500 cases (on a 3MHz dual core Macintosh, OS/X 10.6, 4GB of ram).

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

- All our measures are taken from the test set.
- The $asis$ values are from the neighborhood of the $context$; e.g. the $effort$ column.
- The $tobe$ values are from the cases selected by a treatment; e.g. the $effort$ column.
- 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.
- The $improvement$ from $a = asis$ to $t = tobe$ is $100 * (a - t)/a$. $Larger$ improvements are better.

For example, consider $pmat = 3$:

- Without the $pmat = 3$ restriction, the median and spread in the test set are 235 and 633 months, respectively.
- With $pmat = 3$, the median and spread of projects similar to $context_1$ are 81 and 353 months (see Figure 5).
- The observed improvement in the median is hence 66%.
- The observed improvement in the spread is hence 44%.

3.1 Empirical Example

Figure 7 shows the results seen after apply $\mathcal{W}2$ to:

- Enhancements to a U.K. telecommunications product;
- Projects collected by Miyazaki et al [49];
- Finnish Information Systems projects;
- A large dataset of Chinese software projects;
- Large COBOL projects, collected by Kemerer [50].

The format of this data is highly varied and includes number of basic logical transactions, query count and number of distinct business units serviced. For these data sets, we did not have access to specific case studies like Figure 8. Hence, these results are based on *contexts* developed as follows:

- The first contained the *entire range* of possible project descriptors, representing complete freedom to recommend any change within the space.

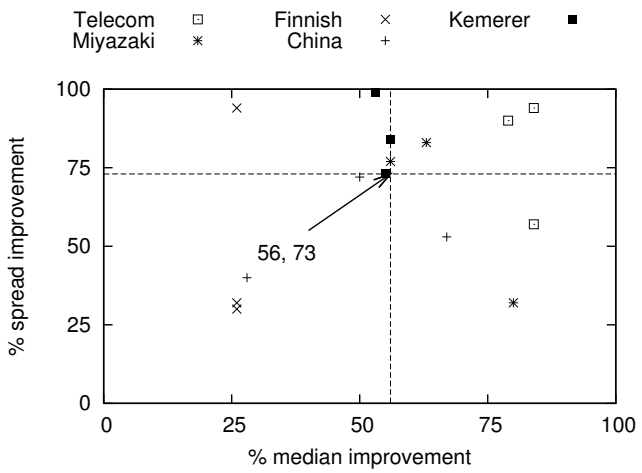


Fig. 7. Effort results for five non-COCOMO datasets.

- 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.

For these experiments, the $values$ function was just "reduce effort" (later in this article we explore other results on COCOMO-related data, that tries to reduce effort *and* defects *and* calendar months).

There are three noteworthy aspects of the Figure 7 results:

- All the points in that figure are positive; i.e. improvements were seen in all cases.
- The dotted lines show the 50% percentile range of the results: half that results had at least 56% and 73% improvement in median and spread.
- There is no evidence that $\mathcal{W}2$ has problems with smaller data sets. The two smallest examples processed by $\mathcal{W}2$ are Kemerer and Telecom containing 15 and 18 examples each. The minimum improvements seen, even for these small data sets, are 55% (in both median and spread).

The expected value of the results in these examples is very high; e.g. a 56% median improvement in effort. The reason for these large improvements is that, in these examples, we focuses *only* on effort. Clearly, there are many ways to cut corners in a project and some of those can have disastrous results (e.g. allocate no effort to testing will reduce the cost, but that is clearly not a recommended management action for a software project). Later in this paper are examples where $\mathcal{W}2$ is chasing improvements to effort *and* defects *and* total calendar time to develop the software. Optimizing for $N = 3$ goals is a harder task than just the $N = 1$ goal of Figure 7. Hence, those the improvements seen in those examples will be more modest (around 20%).

4 Model-based Methods

This section discusses SEESAW, our best model-based method for learning changes to a software project. Subsequent sections will compare SEESAW to $\mathcal{W}$.

4.1 Case Studies

Since all our model-based methods are built around the COCOMO-suite, we must use COCOMO data and *contexts* written in the COCOMO ontology. Figure 8 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 (OSP);
- *OSP2* represents a second, later version of OSP with a more limited scope of COCOMO attributes.

The un*control*able column in Figure 8 shows project features that cannot be changed. For example in project 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 flight software varies from 3 (nominal) to 5 (very high).

4.2 Handling the Models

Optimizing for a set of goals is traditionally resolved by computing partial differential equations of a model, and 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. As shown in Equation 2, this premise does not hold for the COCOMO models used in this study.

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

$$prediction = model(P, T) \tag{4}$$

For example, in local calibration, the tuning options T are the ranges of (a, b) and 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.

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

$$\overbrace{R_x \subseteq control}^{\text{AI search}}, median_n \left(\underbrace{t \subseteq T, value(model(R_x, t))}_{\text{Random Selection}} \right) \tag{5}$$

$median_n$ is the median observed 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):

$$value = 1 - \left(\sqrt{Effort^2 + Defects^2 + Time^2} / \sqrt{3} \right) \tag{6}$$

	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		

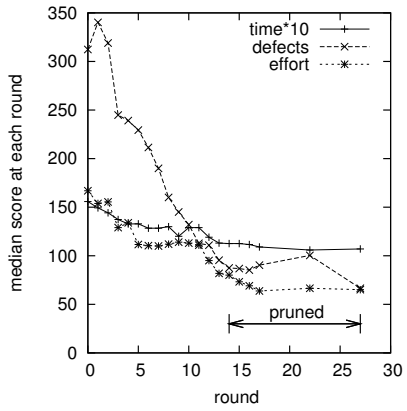
Fig. 8. Contexts of 4 case studies. {*1, 2, 3, 4, 5, 6*} map to {*very low, low, nominal, high, very high, extra high*}.

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

Equation 5 needs to sample the space of model tunings. The appendix to this paper describes one mechanism for sampling across the space of the tunings within Boehm's COCOMO effort estimator and the COQUALMO defect estimator. One thing to note with that appendix is that, for instance-based methods, *none* of that machinery is required.

4.3 Six AI Model-Based Algorithms

The case studies of Figure 8 can be used to assess how well different AI algorithms can find changes to software projects. For example, a typical Simulated Annealing (SA) run explores 10,000 variants on some solution [51]. A side-effect of that run is 10,000 sets of inputs, each scored with the $value$ function



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. 9. Example of SA's forward and back select.

of Equation 6. 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. Figure 9 shows the treatment R_x at any x value is the conjunction of ranges observed between 1 to x (see the table at the bottom of that figure). The y axis scores show 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 [52]. 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 [18] 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, each feature is randomly selected on each iteration. 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 used in SA.

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. 10. Number of times algorithms were top-ranked (largest is 4: i.e. one for each Figure 8 case study).

This paper also explores the ISAMP, BEAM, and A-STAR algorithms described in the appendix. Initially, we planned to explore more AI algorithms but the success of $\mathcal{W}$'s instance-based approach has decreased our motivation in that regard.

4.4 Comparisons of AI Model-based Methods

For each case study of Figure 1, each algorithm was run 20 times (guided by the $value$ function of Equation 6). 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 10 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 10, 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.

5 Model vs. Instance-Based Methods

SEESAW requires models in the COCOMO format so for our comparisons, we restrict ourselves to data in that format. $\mathcal{W}2$ used the historical cases from the NASA93ii and COC81ii datasets. These data sets all have the features defined by Boehm [48]; e.g. analyst capability, required software reliability, and use of software tools. Originally collected in the COCOMO-I format, JPL business experts have translated them from their original COCOMO format to COCOMOII.

Both SEESAW and $\mathcal{W}2$ guided their search using Equation 6 and the four $context$s of §4.1. SEESAW used those $context$s to guide their "what-if" queries around its COCOMO/COQUALMO models. $\mathcal{W}2$ took those $contexts$ then applied the seven step procedure described above to NASA93ii and COC81ii. 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 and $\mathcal{W}2$. We say that $rows_1$ and $rows_2$ are the rows selected from the neighborhood after applying SEESAW's or $\mathcal{W}2$'s recommendations (and by "apply", we mean reject any row that contradicts the ranges in the recommendation). From $rows_i$, we applied Equation 6 to find $values_i$.

The are shown in Figure 11, divided into the defect, effort, months changes see in GROUND, FLIGHT, OSP2 and OSP.

Win	Goal	Treatment	50% percentile (median) $a=$ as is	$t=$ to be	(75-25)th percentile (spread) $A=$ as is	$T=$ to be	Median improv. $\frac{a-t}{a}$	Spread improv. $\frac{A-T}{A}$
NASA93ii Flight								
	defects	SEESAW	1276	626	3737	2311	51%	38%
	defects	W	2042	1688	3992	2501	17%	37%
	effort	SEESAW	159	72	378	192	55%	49%
	effort	W	265	183	416	242	31%	42%
	months	SEESAW	21	15	13	8.6	27%	33%
	months	W	22	20	15	11.1	5%	24%
NASA93ii Ground								
	defects	SEESAW	2006	688	4254	2203	66%	48%
	defects	W	2007	933	3763	1121	54%	70%
	effort	SEESAW	240	95	390	166	61%	57%
	effort	W	177	81	361	156	54%	57%
	months	SEESAW	22	16	15	8.8	28%	41%
	months	W	21	17	14	6.2	19%	55%
NASA93ii OSP								
*	defects	W	1586	767	3557	1741	52%	51%
	defects	SEESAW	1265	1696	3722	3077	-34%	17%
*	effort	W	210	99	557	179	53%	68%
	effort	SEESAW	150	174	411	372	-16%	10%
*	months	W	21	15	15	9.0	28%	39%
	months	SEESAW	21	21	15	12	-2%	21%
NASA93ii OSP2								
*	defects	W	2077	744	4222	1356	64%	68%
	defects	SEESAW	2042	1172	4369	3127	43%	28%
*	effort	W	239	79	465	145	67%	69%
	effort	SEESAW	210	118	514	275	44%	46%
	months	W	21	15	17	6.8	31%	60%
	months	SEESAW	21	16	17	11	25%	36%
COC81ii Flight								
	defects	W	1529	1265	1867	2369	17%	-27%
	defects	SEESAW	1487	1629	2054	1965	-9%	4%
	effort	W	86	81	181	200	6%	-11%
	effort	SEESAW	89	106	246	237	-19%	4%
*	months	W	18	16	6.5	10	11%	-49%
	months	SEESAW	18	20	10	8.9	-8%	8%
COC81ii Ground								
	defects	W	1541	1248	1902	2102	19%	-11%
	defects	SEESAW	1650	1496	2445	2499	9%	-2%
	effort	W	98	65	199	223	33%	-12%
	effort	SEESAW	106	122	383	372	-15%	3%
*	months	W	18	15	9.2	10	17%	-7%
	months	SEESAW	19	19	10	10	0%	-5%
COC81ii OSP								
*	defects	W	1496	1068	1787	2054	29%	-15%
	defects	SEESAW	1496	1765	2233	2233	-18%	0%
	effort	SEESAW	93	83	332	200	11%	40%
	effort	W	88	93	209	205	-5%	2%
*	months	W	19	14	9.0	8.9	22%	1%
	months	SEESAW	19	19	9.4	10	-3%	-4%
COC81ii OSP2								
*	defects	W	1850	1802	2697	2405	3%	11%
	defects	SEESAW	1473	2269	1769	2061	-54%	-17%
*	effort	W	122	130	431	356	-7%	17%
	effort	SEESAW	98	447	289	288	-356%	0%
	months	SEESAW	19	19	7.9	8.6	-3%	-9%
	months	W	20	21	11	10	-4%	10%

Fig. 11. Changes in median and spread.

In all, we show 24 comparisons:

$$\begin{pmatrix} NASA93ii \\ COC81ii \end{pmatrix} * \begin{pmatrix} defects \\ effort \\ months \end{pmatrix} * \begin{pmatrix} ground \\ flight \\ OSP \\ OSP2 \end{pmatrix}$$

$\mathcal{W}2$ produced larger median reductions that SEESAW in $16/24$ comparisons. The "Win" column of those figures indicates when any member of a comparison had a higher value *and* was statistically significantly different (Mann-Whitney, 95% confidence). In nearly half the comparisons ($11/24$), $\mathcal{W}2$ results were statistically different *and* better than SEESAW (in the remaining comparisons, SEESAW's median improvements were never better than $\mathcal{W}2$).

Figure 12 shows the sorts the median and spread improvements seen from the Figure 11 results. Note that rarely were the changes to the median less than zero. In the majority of cases, W2's median and spread improvements were positive (an expected value of 20.5; sometimes ranging over 50%). While occasionally the spread degraded sharply (down to 50% $worse$), such cases were uncommon: note that in only 10% of our results were the spread changes below -15%. Also, all the cases where $\mathcal{W}2$ had poor spread results were in the COC81ii data set which, as discussed below, is a data set with certain special features.

The gray cells in Figure 11 show optimization *failures*; i.e. a zero or negative improvement. $\mathcal{W}2$ failed less than SEESAW (had fewer gray cells). $\mathcal{W}2$ showed $3/24$ and $7/24$ failures for medians and spreads (respectively) while SEESAW showed $13/24$ and $7/24$ failures for medians and spreads (respectively). One of SEESAW's failures was particularly dramatic: witness the increase from 98 effort months to 447 effort months in the OSP2 effort results. We conjecture that SEESAW's greater failures in median reduction are due to the over-fitting problem discussed in §2.4. SEESAW's model-based methods are free to sample increasingly narrow segments of the internal state space of a model ("flying in", as it were, into small cracks between the training data). If that sampling is taken to extreme, and the model-based methods offer recommendations that cover a tiny part of the state space, and if the test data does not fall into that tiny region, then the model-based recommendations will fail.

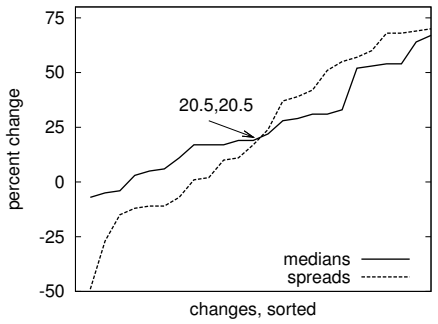


Fig. 12. Range of changes in median and spread generated by applying the recommendations of $\mathcal{W}2$. The median observed changes were (20.5, 20.5)% for (medians, spreads), respectively.

Note that most of the gray cells occur in the COC81ii results. Boehm assumed that this data was to be analyzed by regression so spent much effort on the COC81ii data, applying his domain expertise to prune or trim outstanding values. Curiously, $\mathcal{W}2$ performed best on the "uncleansed" data set (NASA93ii) than the cleaner data set (COC81ii). We conjecture that, sometimes, seemingly "dirty" data actually contains data that is insightful in some contexts. While such outliers confuse regression-based methods (that try to fit one model over the entire data), instance-based tools like $\mathcal{W}2$ can exploit those less-common instances (since they build local models around each context).

In summary:

- $\mathcal{W}2$'s performance was better than SEESAW;
- $\mathcal{W}2$ was more effective at reducing the medians;
- Both instance-based and model-based methods had similar issues with reducing the spread.
- Possibly, the instance-based approach of $\mathcal{W}$ performs better on "dirtier", nosier data than model-based methods.

6 Discussion

6.1 Intra- and Inter-Project Stability

One of the premises of instance-based methods like $\mathcal{W}2$ is that local reasoning in some specific context is best that fitting one model over an entire space. This is required if the "best" solutions in a one context do not hold in others.

To test this premise, we generated a report of what treatments were found under different conditions. Figure 13 shows the results of $\mathcal{W}2$'s $Step7$ (prune all treatments that do appear in less than 50% of 20 repeated trials). The left-hand column of Figure 13 shows the four $values$ function used in that study:

1) *Defects* aims at reducing just defects;
2) *Effort* aims at reducing just effort;
3) *Months* aims at reducing a project's total calendar time.
4) *All* refers to Equation 6; i.e. try to decrease effort *and* development time *and* number of defects;

The last of these is a multi-objective function while the rest strive to optimize one objective without concern to the others. Figure 13 shows that, in any row, the conclusions reached by $\mathcal{W}2$ are stable (i.e. appear at high frequency, across 20 random selections of $train$: $test$). That is, $\mathcal{W}2$'s results exhibit $intra-$project conclusion stability (when the $context$ and $values$ function are held constant). For project managers, this is good news since it shows that their data contains clear signals on how to best change their particular project in order to achieve particular goals.

However, Figure 13 also shows that if the $context$ is changed (from generalized FLIGHT systems to a specific flight system like OSP), then the recommended changes are very different. Similarly, the OSP results show that altering the $values$ function also dramatically changes recommendations.

Menzies & Shull [53] report that many SE papers conclude that what is "best" for one project may not be "best" for another. For example, Zimmermann studied 629 pairs of software development projects [54]. In only 4% of those hundreds of pairs was a defect prediction model learned from one project useful on another. When such *inter*-project conclusion instability exists, then tools like $\mathcal{W}2$ are essential since it is best to learn changes that are tuned to the specifics of particular projects (like OSP & OSP2) rather than on generalized descriptions of software (like FLIGHT & GROUND).

Stability Comparison for NASA93ii FLIGHT

goal	acap=3	apex=3	apex=5	ltex=4	pmat=3	pmat=4	rely=3	sced=2	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 Comparison for NASA93ii GROUND

goal	acap=3	apex=3	apex=5	ltex=4	pmat=3	pmat=4	rely=3	sced=2	stor=3	time=3	changes
defects					80				50		2
effort					60					75	2
months					75		50			75	3
all					85					80	2

Stability Comparison for NASA93ii OSP

goal	acap=3	apex=3	apex=5	ltex=4	pmat=3	pmat=4	rely=3	sced=2	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 Comparison for NASA93ii OSP2

goal	acap=3	apex=3	apex=5	ltex=4	pmat=3	pmat=4	rely=3	sced=2	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. 13. Recommendation frequency across 20 runs of $\mathcal{W}2$ for reducing individual goals ($defects$, $effort$, or $months$) as well as all goals at once (all).

6.2 When Not to Use $\mathcal{W}2$

Like any instance-based method, $\mathcal{W}2$ requires historical cases. If such data is missing then $\mathcal{W}2$ cannot be used.

In that circumstance, discussions about how to best change a project can use results borrowed from other sites. For example, Figure 14 show's Boehm et al.'s [43] analysis of the effects of changing some project attribute from its minimum to maximum value. Based on data from a regression analysis of 161 projects, this figure comments that changing (e.g.) personnel/team capability can alter the effort to build software by up to 350%. Using this data, the effects of various changes can be investigated using Boehm's *delta* analysis technique [55]:

- An old project with known efforts is used as a baseline. A change to a project is described as a new project, expressed in terms of deltas to the variables of Figure 14.
- The new estimate is then the product of the baseline times the effort multiplier deltas.
- The "best" changes to a project are those that are simplest to implement and have most positive impact on the effort (ideally, reduces it).

Column two of Figure 14 lets us compare context-independent reasoning (e.g. delta analysis) vs. context-dependent reasoning

id	appears in Figure 13 as	features	relative weight
1		Personnel/team capability	3.53
2		Product complexity	2.38
3	time	Time constraint	1.63
4	rely	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	apex	Applications experience	1.51
9		Use of software tools	1.50
10		Platform volatility	1.49
11	stor	Storage constraint	1.46
12	pmat	Process maturity	1.43
13	ltex	Language & tools experience	1.43
14	sced	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

Fig. 14. Relative effects on development effort. From [55].

(e.g. $\mathcal{W}2$). Note how that only a third of the Figure 14 attributes appear in the "best" treatments of Figure 13. Curiously, the two attributes with greatest impact (personnel/team capability and product complexity) are absent from Figure 13.

Why is $\mathcal{W}2$ ignoring an attribute with such a large impact (350%)? To answer that question, we have go to the context-dependent particulars. Recall from Figure 8 that in OSP2, product complexity is fixed at $cplx = 4$ and personnel/team capability is fixed at $pcap = 3$. $\mathcal{W}2$ does not recommend treatments for things that cannot change. Hence, $cplx$ and $pcap$ are absent from the OSP2 results of Figure 13. Similarly, OSP allows only $pcap = 3$ so this attribute is also absent.

The same reasoning does not explain the other absent attributes. To understand these, we must look at the data. OSP sets $cplx \in \{5, 6\}$. This attribute is absent in the treatments since there is insufficient support in NASA93ii to justify their inclusion (there only five $cplx = 5$ examples in NASA93ii and no examples of $cplx = 6$). Similar explanations can explain all the remaining absences. Examples such as these show how $\mathcal{W}2$ can provide recommendations that may go against common expert advice. This lack of a defined relationship between data attributes underscores the need for careful query construction. For example, if a query contains conflicting attributes, $\mathcal{W}2$ maintains no internal inconsistency check. Model-based approaches such as S-COST [56] can provide this sanity check, but incur the costs associated with model-based methods discussed above (ontology restrictions, untamed internal model variance, etc).

In summary, when data is absent, managers can debate changes to projects by reusing data like Figure 14. However, the conclusions reached from a context-independent reasoning (like delta analysis) can be made more specific with local information about the kinds of projects seen in the local environment and the kinds of changes the local managers are willing to accept. Therefore, where possible, we recommend collecting local data and analyzing it with $\mathcal{W}2$.

6.3 Scope of the Study

This study use conveniently available datasets in the PROMISE repository, the result applies within the same context of the datasets. In addition, our evaluation compares the performance of different methods across a finite number of problems, so it cannot be used to predict which method will be superior to others for some future, as yet unseen, problem. In fact, no method has been found so far that is universally superior to others in all problems; indeed, the "no-free-lunch theorem" [57] suggests that such an universal best method for all problems can never exist. In practice, for a given new learning problem, various methods need to be empirically evaluated to find the best ones, such as the ones carried out in the study.

We have shown that some treatments identified can improve the quality measures observed in historical project datasets. Our performance measures including median and spread reductions seen in "hold-out data" should not be confused with practical significance in the real world.

That being said, we note that publications from other research communities assess their models in the same manner as this paper: see the *effort estimation* [22]–[24], [30], [31] and *defect prediction* [58]–[61] literature. Ideally, researchers in effort estimation, defect prediction, or learning changes to software projects should apply their recommendations to live projects. However, hold-out tests are widely used due to the tremendous practical difficulties associated with performing such tests on multiple software projects. At the very least, studies like this paper are required to prune the space of methods to be laboriously tested on new, real-world, projects.

7 Conclusion

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

Managers must make management decisions about changes to software projects. Currently, they have only very limited guidance from the SE literature on how to make the fewest number of most effective changes to their projects. Hence, we have spent several years exploring methods for guiding managers towards better choices.

Originally, our work focused on model based methods. Models have many advantages such as representing and visualizing expert domain knowledge. Also, models let us extrapolate from past observations to new situations that may not have been seen previously. As demonstrated by the recent increase in conferences devoted to the construction and exploration of models (e.g. MODELS[1] and SBSE[2]), there is much current interest in model-based software engineering,

Since models seemed so useful, in previous work, we have employed model-based methods to find changes for software projects. This paper reports the surprising result that a small extension to standard instance-based methods (a greedy search

1. http://ecs.victoria.ac.nz/Events/MODELS2011
2. http://www.ssbse.org/2011/

over the neighborhood of some query, divided into a $best$ and $rest$ region) out-performs numerous model-based methods. Specifically, when compared to model-based methods, the $\mathcal{W}2$ instance-based method:

- Is faster to run.
- Is simpler to code;
- Is easier to maintain (just add more cases);
- Is faster to adapt to data sets from new domains;
- Finds equivalent or better ways to improve projects;
- Scales to large problems (since it runs in log-linear time)

Following on from this report, there many issues that could motivate future work:

- Are model-based methods worse for noisier data?
- Is "data cleansing" recommended for regression, but deprecated for instance-based methods?
- How best to reduce spread, thus increase the confidence a user has in the results?
- Why does $\mathcal{W}2$ performs as well as more elaborate model-based methods?

As to this last point, we conjecture that the kinds of process data we can collect from projects are a *shallow source* of knowledge. With such *shallow sources*:

- Very simple methods can plum their depths;
- More elaborate methods may do no better than the very simple.

As evidence for this conjecture, we note that our data sets are often very small (e.g. the 15 rows of KEMERER or the 18 rows of TELECOM). Such small data sources may only hold very limited, and very shallow, structures. If so, then sophisticated AI search algorithms may find little more than the simple greedy search of $\mathcal{W}2$.

This work has focused on software process data. Nevertheless, instance-based methods (such as the $\mathcal{W}2$ algorithm used in this study) can be effectively used in many other fields. For (very long) lists of application areas of instance-based reasoning, see Kolodner [10]. Aamodt [62], Lenz [63], Shepperd [8] and the proceedings of the International Conference on Case-Based Reasoning[3]. While our results *do not* show that *all* instance-based methods are better than *all* instance-based methods, they do motivate more investigations of case-based methods. Before researchers elaborate their model-based methods, it may be both theoretically insightful (as well as pragmatically useful) to build an instance-based version of their method. Based on our experience, we predict that such an instance-based method would be simple to build and, at the very least, provide a baseline against which it is possible to demonstrate the value of more elaborate systems.

REFERENCES

[1] J. Andrews, F. Li, and T. Menzies, "Nighthawk: A two-level genetic-random unit test data generator," in *IEEE ASE'07*, 2007.

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

[3] D. Port, A. Olkov, and T. Menzies, "Using simulation to investigate requirements prioritization strategies," in *IEEE ASE'08*, 2008.

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

3. http://www.iccbr.org/iccbr10/html/program.html

[5] 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.

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

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

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

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

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

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

[12] T. Menzies, "The complexity of trmcs-like spiral specification," in *Proceedings of 10th International Workshop on Software Specification and Design (IWSSD-10)*, 2000.

[13] 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*, 2007.

[14] T. Menzies, O. Elrawas, J. Hihn, M. Feathear, B. Boehm, and R. Madachy, "The business case for automated software engineerng," in *IEEE ASE '07*. New York, NY, USA: ACM, 2007, pp. 303–312.

[15] A. Orrego, T. Menzies, and O. El-Rawas, "On the relative merits of software reuse," in *International Conference on Software Process*, 2009.

[16] T. Menzies, S. Williams, O. El-rawas, B. Boehm, and J. Hihn, "How to avoid drastic software process change," in *ICSE'09*, 2009.

[17] 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, 2009.

[18] T. Menzies, O. El-Rawas, J. Hihn, and B. Boehm, "Can we build software faster and better and cheaper?" in *PROMISE'09*, 2009.

[19] P. Green, T. Menzies, S. Williams, and O. El-waras, "Understanding the value of software engineering technologies," in *IEEE ASE'09*, 2009.

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

[21] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," *IEEE TSE*, vol. 23, no. 12, November 1997.

[22] 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*, 8(2):163-196, 2003.

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

[24] F. Walkerden and R. Jeffery, "An empirical study of analogy-based software effort estimation," *Empirical Softw. Engg.*, 4(2):135-158, 1999.

[25] A. Brady, T. Menzies, O. El-Rawas, E. Kocaguneli, and J. Keung, "Case-based reasoning for reducing software development effort," *JSEA*, December 2010.

[26] A. Brady and T. Menzies, "Case-based reasoning vs parametric models for software quality optimization," in *PROMISE '10*, 2010, pp. 1–10.

[27] A. Ngo-The and G. Ruhe, "Optimized resource allocation for software release planning," *IEEE TSE*, 35(1):109-123, 2009.

[28] 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.

[29] P. Pendharkar, G. Subramanian, and J. Rodger, "A probabilistic model for predicting software development effort," *TSE*, 31(7):615-624, 2005.

[30] 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.

[31] U. Lipowezky, "Selection of the optimal prototype subset for 1-NN classification," *Pattern Recognition Letters*, 19(10):907-918, 1998.

[32] M. Jorgensen and M. Shepperd, "A systematic review of software development cost estimation studies," January 2007.

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

[34] N. E. Fenton and M. Neil, "A critique of software defect prediction models," *IEEE TSE*, vol. 25, no. 5, pp. 675–689, 1999.

[35] M. Harman and J. Wegener, "Getting results from search-based approaches to software engineering," in *ICSE '04*, pp. 728–729.

[36] 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.

[37] A. Miller, *Subset Selection in Regression (second edition)*. Chapman & Hall, 2002.

[38] W. Dillon and M. Goldstein, *Multivariate Analysis: Methods and Applications*. Wiley-Interscience, 1984.

[39] O. Jalali, T. Menzies, and M. Feather, "Optimizing requirements decisions with keys," in *Proceedings of the PROMISE 2008 Workshop*.
[40] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
[41] D. Pfahl and I. Rus, "Special issue on prosim 2004," *Software Process: Improvement and Practice*, vol. 10, no. 3, pp. 251–253, July/September.
[42] T. Menzies, D. Raffo, S. Setamanit, Y. Hu, and S. Tootoonian, "Model-based tests of truisms," in *Proceedings of IEEE ASE*, 2002.
[43] 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.
[44] S. Chulani, B. Boehm, and B. Steece, "Bayesian analysis of empirical software engineering cost models," *IEEE TSE*, 1999, 25(4), 1999.
[45] T. Schulz, L. Radliński, T. Gorges, and W. Rosenstiel, "Defect cost flow model: a bayesian network for predicting defect correction effort," in *PROMISE '10*, 2010, pp. 16:1–16:11.
[46] M. R. Lowry, "Towards predictive models of technology impact on software design productivity," 2010.
[47] 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.
[48] B. Boehm, *Software Engineering Economics*. Prentice Hall, 1981.
[49] 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.
[50] C. Kemerer, "An empirical validation of software cost estimation models," *Comm. of the ACM*, vol. 30, no. 5, pp. 416–429, May 1987.
[51] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, number 220, 4589:671-680, 1983.
[52] 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*, 1993.
[53] T. Menzies and F. Shull, "The quest for convincing evidence," in *Making Software: What Really Works and We We Believe it*, A. Oram and G.Wilson, Eds. O'Reilly Books, 2010.
[54] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction," in *ESEC/FSE'09*, August 2009.
[55] B. Boehm, "Safe and simple software cost analysis," *IEEE Software*, pp. 14–17, September/October 2000.
[56] B. Boehm, B. Boehm, and H. In, "Software cost option strategy tool (s-cost)," in *In Conflict Analysis and Negotiation Aids for Cost-Quality Requirements Annual International Computer Software and Applications Conference), IEEE Comp*. Society Press, 1996, pp. 15–20.
[57] D. H. Wolpert and R. Waters, "The relationship between pac, the statistical physics framework, the bayesian framework, and the vc framework," in *Proc. SFI/CNL Workshop Formal Approaches to Supervised Learning*. Addison-Wesley, 1994, pp. 117–214.
[58] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Where the bugs are," in *ISSTA '04*. New York, NY, USA: ACM, 2004, pp. 86–96.
[59] T. M. Khoshgoftaar and N. Seliya, "Fault prediction modeling for software quality estimation: Comparing commonly used techniques," *Empirical Software Engineering*, vol. 8, no. 3, pp. 255–283, 2003.
[60] A. Tosun, A. Bener, and R. Kale, "Ai-based software defect predictors: Applications and benefits in a case study," in *Twenty-Second IAAI Conference on Artificial Intelligence*, 2010.
[61] T. Menzies, Z. Multon, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, "Defect prediction from static code features: Current results, limitations, new approaches," *Automated Software Engineering*, 2010.
[62] A. Aamodt and E. Plaza, "Case-based reasoning: Foundational issues, methodological variations, and system approaches," *Artificial Intellegence Communications*, vol. 7, pp. 39–59, 1994.
[63] M. L. et. al., *Case-Based Reasoning Technology- From Foundations to Applications*. Springer Verlag, 1998.
[64] 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.

APPENDIX

Modeling Variance in COCOMO/COQALMO

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:

$$\forall x \in \{1..6\}\ EM_i = m_\alpha(x - 3) + 1 \tag{7}$$

where m_α is the effect of α on effort/cost. The *positive effort EM* features such as $cplx$ with slopes m_+, are positively correlated to effort/cost. The *negative effort EM* features such as $acap$, with slopes m_-. are negatively correlated to effort/cost. The m ranges, as seen in 161 projects [55], are:

$$(0.073 \leq m_+ \leq 0.21) \wedge (-0.178 \leq m_- \leq -0.078) \tag{8}$$

To random sample the tunings, all that is required is to select m at random from the ranges of Equation 8. As shown in [16], similar equations can be derived from the COCOMO scale factors and the COQUALMO model.

ISSAMP, BEAM, A-STAR

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 [64]. To avoid exploring low-value regions, our version of ISAMP stores the worst solution observed so far. Any conjunction whose $value$ exceeds that of the worst solution is abandoned, and the new "worst value" is retained. 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 value.

Our other two 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 $1 - Equation\ 6$ (i.e. distance to the utopia of no effort, no development time, and no defects).

Tim Menzies is an 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 master's student at West Virginia University. He received a B.Sc. in Computer Science from WVU in 2009. His current research focuses on instance-based reasoning, specifically its application to improving software quality.

Jacky Keung is an assistant professor at the Department of Computing at the Hong Kong Polytechnic University, and has been working with NICTA on empirical software engineering since 2004. He has a CS degree and a PhD from the University of Sydney and the University of New South Wales respectively and is the author of many top publications. His recent research concerns applying light weight modeling techniques to emerging areas of software technologies, including adaptive elastic cloud computing.

Steven Williams Steven Williams is a PhD student at Indiana University where he studies cognitive science and informatics. He holds BS degrees in civil engineering and computer science from West Virginia University.

Oussama El-Rawas Oussama "Ous" El-Rawas graduated with his Bachelors in Computer Engineering from The American University of Beirut (AUB). He proceeded to obtain his Masters of Science in Electrical Engineering From West Virginia University (WVU) and is currently working at Medquist Inc. as a researcher as well as being a part time PhD student at WVU. His current research interests include Machine Learning, Natural Language Processing, among others.

Phillip Green Phillip Greenn is a Masters student at West Virginia University. His current research focuses on architectural principles for AI toolkits. Currently, he works as a team leader nd lead developer for web-based instructor tools at WVU.

Jairus Hihn Jairus Hihn received the PhD degree in economics from the University of Maryland. He is a principal member of the engineering staff at the Jet Propulsion Laboratory, California Institute of Technology, and is currently the manager for the Software Quality Improvement Projects Measurement Estimation and Analysis Element, which is establishing a laboratory-wide software metrics and software estimation program at JPL. M&Es objective is to enable the emergence of a quantitative software management culture at JPL. He has been developing estimation models and providing software and mission level cost estimation support to JPLs Deep Space Network and flight projects since 1988. He has extensive experience in simulation and Monte Carlo methods with applications in the areas of decision analysis, institutional change, R&D project selection cost modeling, and process models.

Barry Boehm Barry Boehm received his B.A. degree from Harvard in 1957, and his M.S. and Ph.D. degrees from UCLA in 1961 and 1964, all in Mathematics. Between 1989 and 1992, he served within the U.S. Department of Defense (DoD) as Director of the DARPA Information Science and Technology Office, and as Director of the DDR&E Software and Computer Technology Office. He worked at TRW from 1973 to 1989, culminating as Chief Scientist of the Defense Systems Group, and at the Rand Corporation from 1959 to 1973, culminating as Head of the Information Sciences Department. He was a Programmer-Analyst at General Dynamics between 1955 and 1959. His current research interests include software process modeling, software requirements engineering, software architectures, software metrics and cost models, software engineering environments, and knowledge-based software engineering.