Algorithms for Software Quality Optimization

Adam Brady
Lane Department of CS&EE
West Virginia University, USA
adam.m.brady@gmail.com

Jacky Keung
NICTA
University of NSW
Sydney, Australia
Jacky.Keung@nicta.com.au

Tim Menzies
Lane Department of CS&EE
West Virginia University, USA,
tim@menzies.us

Jamie Wood
Lane Department of CS&EE
West Virginia University, USA
jamie.wood.r@gmail.com

Andrei Perhinschi
Lane Department of CS&EE
West Virginia University, USA
andrei.perhinschi@gmail.com

ABSTRACT

Background: There are many data mining methods but few comparisons between them. For example, there are at least two ways to build *quality optimizers*, programs that find project options that change quality measures like defects, development effort (total staff hours), and time (elapsed calendar months). In the first way, we construct a *parametric model* to represent prior software projects. In the second way, we just apply *case-based reasoning* to reason directly from historical cases.

Aim: To assess case-based reasoning vs parametric modeling for quality optimization.

Method: We compared the $\mathcal{W}$ case-based reasoner against the SEEWAW parametric modeling tool.

Results: $\mathcal{W}$ is easy to explain and fast to build. It makes no parametric assumptions and hence can be rapidly applied to project data in many formats. SEESAW is an elaborate tool that can only process project data expressed in a particular ontology (i.e. just the COCOMO attributes). It is also slower to execute than $\mathcal{W}$. In 24 different tests comparing $\mathcal{W}$ and SEESAW, $\mathcal{W}$ always performs at least as well as SEESAW. In 6 of those tests $\mathcal{W}$ performed statistically better (all tests used Mann-Whitney, 95% confidence). Lastly, like any CBR method, it comes with a built-in maintenance strategy (just add more cases).

Conclusion: The $\mathcal{W}$ case-based reasoning tool is recommended over the SEESAW parametric modeling tool (except in the case where there is no local data).

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*Software quality assurance*; D.2.9 [**Software Engineering**]: Management—*Time Estimation*; I.2.6 [**Artificial Intelligence**]: Learning—*Analogies*

Keywords

Effort Estimation, Analogy, Optimization, Parametric modeling, Software Quality, COCOMO, Case Based Reasoning

1. INTRODUCTION

How should we reason about software projects? Should we extrapolate from old data to build a parametric model; e.g. using a Bayes net [9], or the linear equations of COCOMO [5,7]? Or is it best to reason directly from data, without an intervening parametric model, using case-based reasoning (CBR) [26]?

This is a difficult question to answer, unless we restrict ourselves to a particular context. In this paper, we adopt the context of *software quality optimization*; i.e. adjusting a software project such that we improve quality attributes such as the *defects* (number of delivered defects), the *months* (calendar time to delivery) and the *effort* (staff time, in person months, required for that delivery). This *quality optimization* task is different to *effort estimation*. Effort estimators just predict measures on the *current* project while quality optimizers seek *changes* that most improve a project.

Quality optimization is a non-linear problem. Improving any one goal can harm the others. For example:

- If management rushes projects to completion, they decrease *months* but can increase *defects*.
- Projects that adopt elaborate quality assurance procedures can reduce *defects* but at the cost of increased *effort*.

A quality optimizer must therefore trade-off between reducing months *and* defects *and* effort. This paper will compare two quality optimizers:

1. SEESAW is an AI algorithm that explores parametric models of software development, based on COCOMO.
2. $\mathcal{W}$ is a case-based reasoning algorithm that does the same task as SEESAW, without using a parametric model.

SEESAW was first introduced in [18] and has been applied to numerous domains [10,16,17,20–22]. $\mathcal{W}$ was first introduced in [8] but that report includes no comparisons with other quality optimizers. This paper compares $\mathcal{W}$'s case-based reasoning against SEESAW's parametric models. Compared to SEESAW:

- $\mathcal{W}$ finds similar or better optimizations.
- $\mathcal{W}$ is simpler to code: 200 lines of AWK as opposed to the 5000 lines of LISP code used in SEESAW.
- $\mathcal{W}$ is faster to run: the following experiments took minutes for $\mathcal{W}$, but hours for SEESAW.
- $\mathcal{W}$ is simpler to maintain since, in CBR, "maintenance" means nothing more than "add more cases".
- $\mathcal{W}$ makes no use of an underlying model and is therefore free of all the assumptions of parametric modeling. Hence it can be quickly applied to more data sets. For example, SEESAW

requires data to be in the COCOMO format but $\mathcal{W}$ has been applied to numerous data sets in other formats [8].

We conclude from these results that, for the task of quality optimization, $\mathcal{W}$'s case-based reasoning methodology is recommended over SEESAW's parametric modeling. The one exception to this would be that if there is no local data, then $\mathcal{W}$ cannot function and SEESAW should be used. While we offer no conclusion on the general merits of case-based reasoning compared to parametric modeling, these results should encourage further experimentation on the matter.

2. BACKGROUND

The debate between case-based reasoning and model-based methods can be conducted on at least two levels:

1. At one level, it is an engineering-based discussion that assesses these approaches on criteria like ease of implementation, runtime speed, and the observed output performance.
2. At another level of assessment, we can assess case-based vs model-based in terms of their cognitive implications.

Since most of this paper is about level (1), the rest of this section discusses level (2).

Platonic model-based reasoning is meant to seek out universal truths. For example, Newton's agenda was to find a set of equations (e.g. $F = ma$) that can be applied universally on earth, as well as to well as distant planets and stars. He succeeded. In 1846, rival astronomers John Adams (in England) and Urbain Leverrier (in France) raced to find a previously unseen planet that was disturbing the orbit of Uranus. Neptune was first sighted by Adams, then Leverrier, after both men pointed their telescopes at the precise point in the sky indicated by Newton's equations.

We dream of the day that our SE models will achieve the same universality of Newton's equations. To date, we have not been successful. Researchers like Boehm developed parametric models that predict development effort for software. In Boehm's COCOMO parametric model (the 1981 version [5]):

$$Effort = a * Loc^b * \prod_i \beta_i x_i \tag{1}$$

where x_i are one of the *effort multipliers* shown in Figure 1 (at top) and β_i is a coefficient that controls the influence of x_i.

Such learning combines expert intuition with automatic reasoning. Expert intuitions define the general form of the parametric model, while automated data mining fills in the details of that model. For example, the goal of data mining over parametric models is to take local data and learn appropriate values for the tunable attributes. In the above model, those tunable attributes are (a, b, β_i).

Based on linear regression over historical data [5,7], Boehm offers values to (a, b, β_i) to three significant figures. Previously [15], we have reported that such precision is somewhat optimistic since β_i has a very large variance. The plot at the bottom of Figure 1 shows the β_i values learned from twenty 66% samples (selected at random) of the NASA93 data set from the PROMISE repository. While some of the coefficients are stable (e.g. the white circles of *loc* remains stable around 1.1), the coefficients of other attributes are highly unstable:

- The $(max - min)$ range of some of the coefficients is very large; e.g. the upside down black triangles of *stor* ranges from $-2 \leq \beta_i \leq 8$.
- Consequently, nine of the coefficients in Figure 1 jump from negative to positive.

upper: in theory $\beta < 0$	acap: analysts capability
	pcap: programmers capability
	aexp: application experience
	modp: modern programming practices
	tool: use of software tools
	vexp: virtual machine experience
	lexp: language experience
middle	sced: schedule constraint
lower: in theory $\beta > 0$	data: data base size
	turn: turnaround time
	virt: machine volatility
	stor: main memory constraint
	time: time constraint for cpu
	rely: required software reliability
	cplx: process complexity

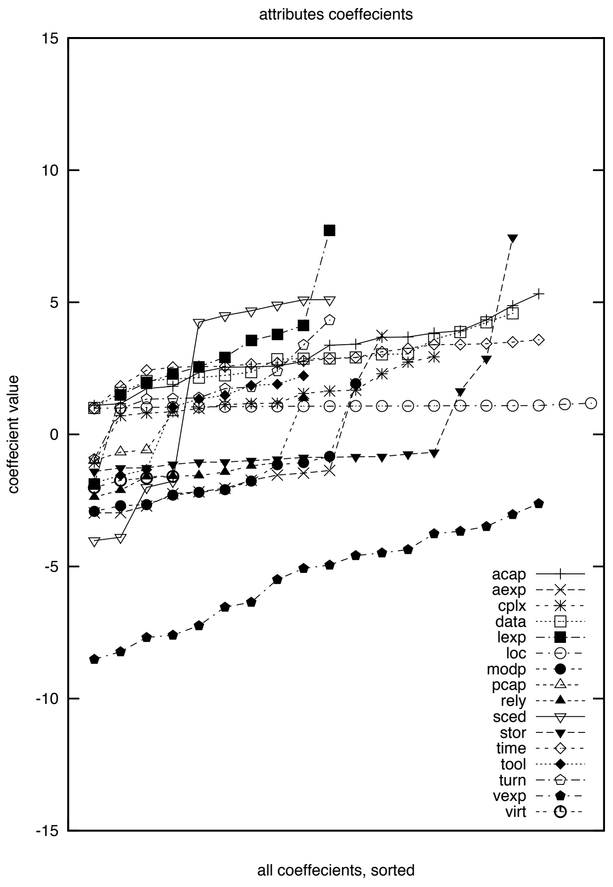


Figure 1: COCOMO 1 effort multipliers, and the sorted coefficients found by linear regression from twenty 66% sub-samples (selected at random) from the NASA93 PROMISE data set; from [15]. Prior to learning, training data was *linearized* in the manner recommended by Boehm (x was changed to $log(x)$; for details, see [15]). During learning, a greedy back-select removed attributes with no impact on the estimates: hence, some of the attributes have less than 20 results. After learning, the coefficients were unlinearized.

We have seen instability in other datasets, including the COC81 data used by Boehm to derive Equation 1 [15]. This is an troubling observation. It seems that while Newton's equations let us precisely locate Neptune, Boehm's equations cannot point us exactly at which project attributes will lead to lower effort.

Parametric modeling assumes that (i) one parametric form (e.g. Equation 1) is universal across multiple domains and (ii) that form is tuned to the local situation by adjusting some tuning attributes.

An opposite approach to parametric models is case-based reasoning (CBR). In CBR, there are no universally-applicable parametric models. Rather, every conclusion is dependent on the particulars of the task at hand. CBR is based on a theory of *reconstructive memory*. According to this theory, humans do not remember things as they actually happened. Rather, "remembering" is an inference process, characterized by Bartlett as:

> *... a blend of information contained in specific traces encoded at the time it occurred, plus* (retrieval time) *inferences based on knowledge, expectations, beliefs, and attitudes derived from other sources [4].*

Bartlett's work was ignored when first published (1932) but today it is highly influential; e.g. experts in psychology & law caution reconstructive memory means that *leading questions* can significantly alter a report given by a human witness [14].

In AI research, Janet Kolodner [13] used reconstructive memory to characterize expert explanations. To support her claim, she offered a set of transcripts of experts explaining some effect. Her reading of those transcripts was that the experts do not use *verbatim recalling* when discussing the past. Rather, they *reconstruct* an account of their expertise, on the fly, in response to a particular query. CBR inference is usually characterized [1] in four steps:

1. *Retrieve*: Find the most similar cases to the target problem.
2. *Reuse*: Adapt our actions conducted for the past cases to solve the new problem.
3. *Revise*: Revise the proposed solution for the new problem and verify it against the case base.
4. *Retain*: Retain the parts of current experience in the case base for future problem solving.

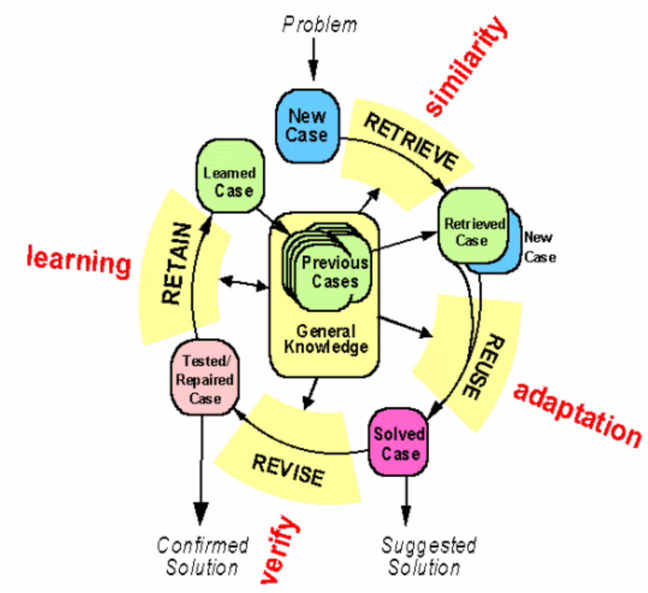


Figure 2: Four steps of CBR, from `http://www.peerscience.com/intro_cbr.htm`.

Having verified the results from our chosen adapted action on the new case, the new case is added to the available case base. The last step allows CBR to effectively learn from new experiences. In this manner, a CBR system is able to automatically maintain itself.

In terms of cognitive theory, CBR challenges notions of reasoning as model-building. The mantra of CBR is "don't think, remember". That is, when faced with some new situation:

- Do not reason it out using some underlying model (e.g. Newton's equations or Boehm's parametric models).
- Rather, respond to a new situation via an on-demand survey of past experiences [23].

CBR challenges the premise of the PROMISE conference series. Currently, this conference bills itself as "Predictive Models in Software Engineering". This title assumes that model building is the best way to analyze software engineering. However, if *model-heavy* methods like COCOMO do worse than *model-lite* CBR methods, then we would need to rethink the premise of PROMISE.

(Note that we call CBR *model-lite*, but not *model-free*. For more on this distinction, see the *Discussion* section, below.)

3. QUALITY OPTIMIZATION

The above discussion motivates a comparison between parametric model-based methods and CBR. To make that comparison, we need to explore the same task with two different approaches. Accordingly, this section describes quality optimization using SEESAW's parametric models or $\mathcal{W}$'s case-based reasoning.

One thing that may not be apparent from the following discussion is the relative complexity of the two systems. Based on recent experience with teaching graduate AI, we assert that building and assessing SEESAW is a term project while $\mathcal{W}$ can be implemented in two weekly homework assignments (week1 implements some basic data loading and nearest neighbor measures; week2 extends that code to complete $\mathcal{W}$).

3.1 SEESAW

Since 2007, we have applied AI algorithms over parametric models of software development (based on COCOMO) [18] to implement quality optimizers. We found this to be a challenging task since it must execute over partial descriptions of projects and, in the case of parametric models, over models with uncertain internal parameters (like the ranges shown in Figure 1).

In order to address this challenge, we need to understand the nature of those models. In parametric modeling, the predictions of a model about a software engineering project are altered by project variables P and *tunable* attribute coefficients T:

$$prediction = model(P, T) \tag{2}$$

In the simplified COCOMO model of Equation 3, the tuning options T are the range of (a, b) and the project options P are the range of *pmat* (process maturity) and *acap* (analyst capability).

$$effort = a \cdot LOC^{b+pmat} \cdot acap \tag{3}$$

Based on the definitions of the COCOMO model we can say that the ranges of the project attributes are $P = 1 \leq (pmat, acap) \leq 5$. Further, given the cone of uncertainty associated with a particular project p, we can identify the subset of the project options $p \subseteq P$ relevant to a particular project. For example, a project manager may be unsure of the exact skill level of team members. However, if she were to assert "my analysts are better than most", then p would include $\{acap = 4, acap = 5\}$.

SEESAW seeks a treatment $r_x \subseteq p$ that maximizes the *value* of a model's predictions where *value* is a domain-specific function that scores model outputs according to user goals:

$$\arg\max_x \left(\overbrace{r_x \subseteq p}^{AI\ search}, \underbrace{t \subseteq T, value(model(r_x, t))}_{Monte\ Carlo} \right) \tag{4}$$

The intuition of Equation 4 was that, when faced with tuning variance like that seen in Figure 1, we should search for conclusions that are stable across the space of possible tunings. SEESAW assumed that the dominant influences on the $prediction$ are the project options p (and not the tuning options T). Under this assumption, the predictions can be controlled by:

- Constraining p (using some AI tool)
- Leaving T unconstrained (and sampling $t \in T$ using Monte Carlo methods)

The parametric models used by SEESAW's models come from COCOMO. These attributes have a range taken from {very low, low, nominal, high, very high, extremely high} or

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

In COCOMO-II model [7], Boehm divided the attributes into two sets: the *effort multipliers* and the *scale factors*. The effort multipliers affect effort/cost in a linear manner. Their off-nominal ranges {vl=1, l=2, h=4, vh=5, xh=6} change the prediction by some ratio. The nominal range {n=3}, however, corresponds to an effort multiplier of 1, causing no change to the prediction. Hence, these ranges can be modeled as straight lines $y = mx+b$ passing through the point (x, y)=$(3, 1)$. Such a line has a y-intercept of $b = 1-3m$. Substituting this value of b into $y = mx + b$ yields:

$$\forall x \in \{1..6\}\ EM_i = m_\alpha(x - 3) + 1 \tag{5}$$

where m_α is the effect of α on effort/cost.

We can also derive a general equation for the scale factors that influence cost/effort in an exponential manner. These features do not "hinge" around (3,1) but take the following form:

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

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

Along with COCOMO-II, Boehm also defined the COQUALMO defect predictor. COQUALMO contains equations of the same syntactic form as Equation 5 and Equation 6, but with different coefficients. Using experience from 161 projects [7], 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 4), all we need to do is select m values at random from the min/max m values ever seen. An appendix to this document lists those ranges.

Initially, we implemented the AI search of Equation 4 using simulated annealing [17, 18, 21]. Subsequent work demonstrated that the recommendations found in this way did better than numerous standard process improvement methods [20]. Later implementations were based on a state-of-the-art theorem prover [10]. SEESAW searches within the ranges of project attributes to find constraints that most reduce development effort, development time (measured in calendar months), and defects. Figure 3 shows SEESAW's pseudo-code. The code is an adaption of Kautz & Selman's MaxWalkSat local search procedure [13]. The main changes are that each solution is scored via a Monte Carlo procedure (see score in Figure 3) and that SEESAW seeks to minimize that score (since, for our models it is some combination of defects, development effort, and development time in months).

```
1 function run (AllRanges, ProjectConstraints) {
2   OutScore = -1
3   P = 0.95
4   Out = combine(AllRanges, ProjectConstraints)
5   Options = all Out features with ranges low < high
6   while Options {
7     X = any member of Options, picked at random
8     {Low, High} = low, high ranges of X
9     LowScore = score(X, Low)
10    HighScore = score(X, High)
11    if LowScore < HighScore
12      then Maybe = Low; MaybeScore = LowScore
13      else Maybe = High; MaybeScore = HighScore
14    fi
15    if MaybeScore < OutScore or P < rand()
16      then delete all ranges of X except Maybe from Out
17      delete X from Options
18      OutScore = MaybeScore
19    fi
20  }
21  return backSelect(Out)
22 }
23 function score(X, Value) {
24   Temp = copy(Out) ;; don't mess up the Out global
25   from Temp, remove all ranges of X except Value
26   run monte carlo on Temp for 100 simulations
27   return median score from monte carlo simulations
28 }
```

Figure 3: Pseudocode for SEESAW

SEESAW first combines the ranges for all project attributes. These constraints range from Low to High values. If a project does not mention a feature, then there are no constraints on that feature, and the combine function (line 4) returns the entire range of that feature. Otherwise, combine returns only the values from Low to High. In the case where a feature is fixed to a single value, then Low = High. Since there is no choice to be made for this feature, SEESAW ignores it. The algorithm explores only those features with a range of Options where Low < High (line 5). In each iteration of the algorithm, it is possible that one acceptable value for a feature X will be discovered. If so, the range for X is reduced to that single value, and the feature is not examined again (line 17). SEESAW prunes the final recommendations (line 21). This function pops off the N selections added last that do not significantly change the final score (t-tests, 95% confidence). This culls any final irrelevancies in the selections. The score function shown at the bottom of Figure 6 calls COCOMO/COQUALMO models 100 times, each time selecting random values for each feature Options. The median value of these 100 simulations is the score for the current project settings. As SEESAW executes, the ranges in Options are removed and replaced by single values (lines 16-17), thus constraining the space of possible simulations.

While a successful prototype, SEESAW has certain drawbacks:

- *Model dependency:* SEESAW requires a model to generate the estimates. Hence, the conclusions reached were only as good as this model so using this tool requires an initial, possibly time-consuming, model validation process.
- *Data Dependency:* SEESAW can only process project data in a format compatible with the underlying model. In practice, this limits the scope of the tool.
- *Arbitrary Design*: SEESAW handles two dozen cases using rules designed using "engineering judgment"; i.e. they are not based on any theoretical or empirical results in the literature (for example, "do not increase automatic tools usage without increasing analyst capability"). The presence of such ad hoc rules makes it harder to verify that the tool is correct.
- *Performance*: SEESAW uses tens of thousands of iterations,

with several effort estimates needed calculated for each iteration. This resulted in a performance disadvantage.
- *Size and Maintainability*: Due to all the above factors, the SEESAW code base has proved difficult to maintain.

We have found that these factors limit the widespread use of quality optimizers:

- In the three years since our first paper [18], we have only coded one software process model (COCOMO), which inherently limits the scope of our investigations.
- No other research group has applied these techniques.

These problems motivated an exploration of alternate approaches to quality optimization.

3.2 Contrast Set Learning

The cognitive basis of case-based reasoning offers an alternative to model-based approaches for software quality estimation. One can relate the use of a model as an attempt to extrapolate a single, general, *verbatim recollection* of knowledge in order to infer about any related problems. For example, consider the case of Brooks' law. Brooks' law states that adding more programmers to an already delayed project will only delay it further. Thus, a general, model-based approach for software quality would state:

Do not add programmers to an already delayed project if schedule deadlines are important.

These general guidelines are similar to the scale factors and effort multipliers of the USC COCOMO model 1. In the case of COCOMO, for example, very low analyst capability (*pcap*) always signals a shift towards greater project effort and lower software quality. By combining these guidelines into a general model as Boehm has done, we can use tools like SEESAW to optimize software quality given a constrained query on the software project considered.

Because of the many issues regarding SEESAW's model-based approach already outlined, we have devised a modification of CBR that attempts to instead *reconstruct* smaller, local accounts of what drives better software quality. Using CBR as a guide, we seek a way to *retrieve* relevant information about a software project, *reuse* that information to extract possible improvements, *revise* our proposed quality estimate, then *retain* this knowledge for future estimates.

We refer to this simple process as Contrast Set Learning. CSL seeks to optimize the choice of potential actions based solely on what how similar, historical instances implementing those actions have performed. In terms of software quality: given a manager's possible decisions and accounts of past decisions, which decisions are most likely to improve software quality?

CSL extracts these decisions using *contrast sets*. From the pool of historical cases, those most similar to the project at hand are selected. From this local neighborhood, cases are sorted based on some utility measure such as software effort. Two sets are formed, one labelled "best" that contains the most desireable cases, and one labelled "rest" that contains the remaining cases. The contrast set consists of the attributes that occur more often in "best" than "rest". That is, attributes that occur frequently in only one population tell us more than attributes that appear in both. Such attributes provide more *contrast* between the populations. That is, high contrast attributes better encapsulate the properties that separate the two populations, whereas attributes with low contrast tell us very little about the space as a whole.

3.2.1 A Worked Example of CSL

For example, consider a company faced with a software project that is sprialling out of control. Currently they are missing deadlines, lack funding, and lack programmers. Management decides to solve this problem by hiring additional programmers. Not surprisingly, in accordance with Brooks' law, the additional training time causes the project to be delivered even later, and now grossly over budget.

Using contrast set learning, management would instead look at past projects from the same or similar departments. Say the best projects (those requiring the least effort) were either given additional funds or allowed to extend their deadlines $best = (more - money|extend - deadlines)$. Now consider that the remaining projects (those requiring the most effort) were either given additional programmers or allowed to extend their deadlines $rest = (more - programmers|extend - deadlines)$.

Looking at these decisions we can infer little effect from extending deadlines, as this change occurred equally in both the $best$ and $rest$ sets. Thus, it provides little contrast. However, both adding funding and adding programmers present *high-contrast*. Only the best projects involved adding money, and only the worst projects involved adding programmers.

3.3 $\mathcal{W}$

The standard procedure for CBR is to report the median class value of some local neighborhood. This neighborhood is typically defined as the Euclidean distance from a defined project in n-dimensional space with n project features [26]. $\mathcal{W}$ works similarly, but defines a project as a range of values:

- From a range of project values, cases are retrieved that match a specific amount of *overlap* with the defined project ranges. A case's overlap is defined as the percentage of attributes that fall within the specified ranges of the defined project.
- From these selected similar cases, the cases are sorted by a measure of utility to determine the *better* examples.
- From these sorted ranges, a *contrast set* is learned. The top 5 "best" cases (those with the best utility measure) are placed into a set labeled "best". The next 15 ranked cases are placed into a set labeled "rest", for a combined total of 20 cases.
- From the contrast set, $\mathcal{W}$ selects the features that best select for the region with the best utility measurements.

In the above, *better* is determined by some domain-specific predicate. In the case of effort, defect, and month estimations, this utility is the normalized euclidean distance from the lowest possible cost for all three factors.

3.3.1 Contrast Sets

Once a contrast set learner is available, it is a simple matter to add $\mathcal{W}$ to CBR. $\mathcal{W}$ finds contrast sets using a greedy search, where candidate contrast sets are ranked by the frequency of which they appear in the "best" set squared divided by how often the candidate appears in both the "best" and "rest" sets. A simple strategy to score more favorably towards attributes that occur most often in the best case is to square the number of times. Taking this heuristic one step further, given an attribute x, we can penalize x's occurrence in the "rest" by dividing the sum of the frequency counts in best and rest [16], the ensuring rare attributes are weighted appropriately:

$$like = \frac{freq(x|best)^2}{freq(x|best) + freq(x|rest)} \tag{7}$$

From this measure we need only sort each attribute by it's $like$ score to prioritize our recommendations

```
@project example
@attribute ?rely 3 4 5
@attribute tool  2
@attribute cplx 4 5 6
@attribute ?time 4 5 6
```

Figure 4: $\mathcal{W}$'s syntax for describing the input query q. Here, all the values run 1 to 6. $4 \leq cplx \leq 6$ denotes projects with above average complexity. Question marks denote what can be controlled- in this case, $rely, time$ (required reliability and development time)

3.3.2 The $\mathcal{W}$ Algorithm

CBR systems input a query q and a set of cases. They return the subset of cases C that is relevant to the query. In the case of $\mathcal{W}$:

- Each case C_i is a historical record of one software project, plus the development effort required for that project. Within the case, the project is described by a set of attributes which we assume have been discretized into a small number of discrete values (e.g. analyst capability $\in \{1, 2, 3, 4, 5\}$ denoting very low, low, nominal, high, very high respectively).
- Each query q is a set of constraints describing the particulars of a project. For example, if we are interested in a schedule over-run for a complex, high reliability project that has only minimal access to tools, then those constraints can be expressed in the syntax of Figure 4.

$\mathcal{W}$ seeks q' (a change to the original query) that finds another set of cases C' such that the median effort values in C' are less than that of C (the cases found by q). $\mathcal{W}$ finds q' by first dividing the data into two-thirds training and one-third testing. *Retrieve* and *reuse* are applied to the training set, then *revising* is applied to the test set.

In the *retrieve* step, the initial query q is used to find the N training cases nearest to q using a Euclidean distance measure where all attribute values are normalized from 0 to 1.

In the *reuse* (or adapt) step, the N cases are sorted by effort and divided into the K_1 best cases (with lowest efforts) and K_2 rest cases. For this study, we used $K_1 = 5, K_2 = 15$. Then we seek the contrast sets that select for the K_1 best cases with the *better* estimates. All the attribute ranges that the user has marked as "controllable" are scored and sorted by Equation 7. This sorted order S defines a set of candidate q' queries that use the first *i-th* entries in S:

$$q'_i = q \cup S_1 \cup S_2 ... \cup S_i$$

According to Figure 2, after *retrieving* and *reusing* comes *revising* (this is the "verify" step). When revising q', $\mathcal{W}$ prunes away irrelevant ranges using the algorithm of Figure 5.

On termination, $\mathcal{W}$ recommends changing a project according to the set $q' - q$. For example, in Figure 4, if $q' - q$ is $rely = 3$ then this treatment recommends that the best way to reduce the effort for this project is to reject $rely = 4 \; or \; 5$.

Formally, the goal of $\mathcal{W}$ is find the smallest i value such that q'_i selects cases with the more of the *better* estimates. The reader might protest that the generation of some succinct human-readable construct like q'_i means that $\mathcal{W}$ is not a "real" case-based reasoner. In that view, the distinguishing feature of CBR is that its reasoning is instance-based and it never generates any generalizations.

In reply, we observe that $\mathcal{W}$ is not the only system that extends standard CBR with some generalization tools. Watson [27] reviews numerous CBR systems that, for example, run decision tree learners over their case library in order to automatically generate an index to the cases. Also, once a system can read a case library, compute distance calculations, and generate a sorted list of the nearest neighbors, implementing Figure 5 and Equation 7 is only a few dozen lines of code. That is, $\mathcal{W}$ is such a small extension to standard CBR that it would be somewhat pedantic to declare that it is not "real" CBR.

```
1. Set $i = 0$ and $q'_i = q$
2. Let $Found_i$ be the test cases consistent with $q'_i$ (i.e. that do not
   contradict any of the attribute ranges in $q'_i$).
3. Let $Effort_i$ be the median efforts seen in $Found_i$.
4. If $Found$ is too small then terminate (due to over-fitting). After
   Shepperd [26], we terminated for $|Found| < 3$.
5. If $i > 1$ and $Effort_i < Effort_{i-1}$, then terminate (due to
   no improvement).
6. Print $q'_i$ and $Effort_i$.
7. Set $i = i + 1$ and $q'_i = q_{i-1} \cup S_i$
8. Go to step 2.
```

Figure 5: Revising q to learn q'.

4. THE $\mathcal{W}$2 ALGORITHM

Upon initial experimentation with $\mathcal{W}$, we were forced to decide upon a few arbitrary values for internal decisions. For example, when deciding which historical cases were relevant to a given project, we chose the standard CBR method of taking k nearest neighbors based on euclidean distance from the defined query. Given the size of our datasets we arbitrarily chose $k = 20$ for our definition of the closest neighbors.

This proved problematic in two regards. First, the knn calculation required $O(n^2)$ time to run, limiting our application to very large datasets. Second, the arbitrary selection of 20 cases (separated into the 5 "best" and 15 "rest") often selected too large a subset of the data for certain datasets. For example, if data was only provided for 12 historical cases, once separated into a training set of 66%, only 8 cases remain. At this point no relevancy filtering is performed, and the entire space is selected for learning.

To resolve this, a non-static metric for relevancy was devised. Instead of selecting cases based on an arbitrary value, cases were ranked and selected based on how well they were contained within the query space. For each attribute in a case, the case was compared to the project query. If the case's value falls within the query, the case scores on "point" for that attribute. These scores are combined and ranked. For example, if a case within the $nasa93$ dataset falls within the Orbital Space Plane (OSP) case study query for 16 of its attributes and fails for the other 7, it is said to be 70% contained. The cases with the highest containment ("Best Overlap") are then selected for contrast set reasoning.

The performance of this new method is shown in figure 7. KNN represents the old $O(n^2)$ method of relevancy filtering compared with the new BestOverlap method. In all but one case, BestOverlap performs better. However, even when BestOverlap performs slightly worse, it still performs better than KNN in spread reduction.

5. COMPARING $\mathcal{W}$ TO SEESAW

In order to compare $\mathcal{W}$ and SEESAW, both systems require similar inputs. SEESAW can only handle models in the COCOMO format. Hence, we restrict ourselves to data in that format (see [8] for examples of $\mathcal{W}$ running on a much broader set of inputs).

The inputs required for this study are:

	Execution Time		
dataset	W	W2	W2 speedup
nasa93	0.69s	0.10s	6.6x
coc81	0.43s	0.08s	5.3x
china	0.37s	0.42s	10.8x
telecom1	0.07s	0.04s	1.6x

Figure 6: Average execution times for the W and W2 algorithms. By removing the $O(n^2)$ kth nearest neighbor calculation from W we drastically improve performance, especially on larger datasets such as China (500 instances).

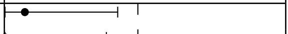
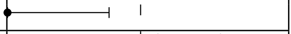
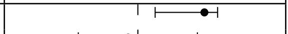
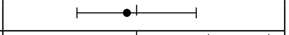
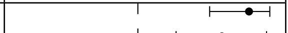
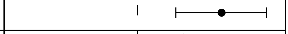
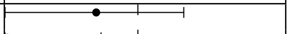
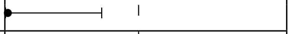
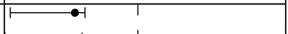
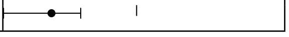
Dataset	Treatment	Median Reduc	Spread Reduc	Reduction Quartiles 50%
kemerer	BestOverlap	7%	48%	
kemerer	KNN	0%	44%	
miyazaki*	BestOverlap	75%	24%	
miyazaki	KNN	46%	45%	
telecom1	KNN	92%	23%	
telecom1	BestOverlap	81%	34%	
china	BestOverlap	34%	67%	
china	KNN	1%	36%	
finnish	BestOverlap	26%	28%	
finnish	KNN	18%	29%	

Figure 7: Performance of W2's BestOverlap relevancy filtering vs W's kth nearest-neighbor filtering for 5 unique datasets.

- $\mathcal{W}$ needs a set of *historical cases*. We used the NASA93 dataset available from `http://promisedata.org/data`. This dataset represents 93 different NASA projects collected from the 1980's and 1990's represented as feature vectors describing each project in COCOMO format. NASA93 data only contains historical information for project effort. Development time (measured in calendar months) and defects were added in using the COCOMO/COQUALMO models.
- Both SEESAW and $\mathcal{W}$ need an *objective function* that guides their search. In this study, the objective function rewarded minimization of the sum of defects and effort and months (after these values had been normalized to the same range).
- Both SEESAW and $\mathcal{W}$ need a set of *project constraints* that tune their conclusions to particular projects. We used the project constraints of Figure 8.

Figure 8 comes from our debriefing of NASA program managers and shows different kinds of NASA mission:

- *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 $values$ column in that figure shows settings 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 recommendations for that project. For instance, the reliability of the JPL flight software can vary from a ranking of 3 (nominal) to 5 (very high).

$\mathcal{W}$ used Figure 8 to set its initial query q_0. SEESAW used Figure 8 to guide a set of simulations around its parametric models. For each case study, 1000 times, inputs were selected at random, constrained by Figure 8 (so the inputs for case study X conformed to the description of X shown in that figure).

In order to offer a fair comparison between SEESAW and $\mathcal{W}$, we proceeded as follows. Recall that $\mathcal{W}$ has a *training* component that implements *retrieve*, *reuse*, and *revise* (described around Figure 5). A *test* component was implemented that copied the code used for *retrieve*. This test component was modified such that it executed on a different *test set* that contained no data used in *training*.

Given that rig, for each case study in Figure 8, we repeated the following process 50 times.

- The available data (NASA93) was divided into a *train* and *test* sets (of sizes 66%:33%). The division was random so that each time, different instances appeared in train and test.
- The median and spread values for effort, months, and defects were collected from the *train set*. These medians and spreads were recorded as the $before$ values.
- Each quality optimizer ($\mathcal{W}$ and SEESAW) was run separately. The $\mathcal{W}$ algorithm used the *train set* while SEESAW used its internal models. In either case, the quality optimizer returned a set of recommendations on how to change the project in order to reduce effort, defects, and development time (measured in calendar months).
- These recommendation were assessed in the same way: by passing them to $\mathcal{W}$'s test component which retrieved relevant cases from the *test set*.
- The median and spread values for effort, months, and defects were collected from the instances retrieved from the *test set*. These were recorded as the $after$ values.

The results were reported in terms of *median* and *spread*. We say that the $median$ of a set of numbers are the 50th-percentile value while the $spread$ is difference between the 75th and 25th percentile value. The median is a measure of central tenancy while the spread is a measure of uncertainty around the median. Decreasing the spread means that the predictions fall within a narrower range. We report spread rather than other measures like standard deviation since we wish to avoid any inappropriate assumptions of symmetrical distributions.

6. RESULTS

6.1 $\mathcal{W}$ vs SEESAW

Average median and spread results over the 50 trials are shown in Figure **??**. The last column in each group (labeled "Change") shows the relative change in effort, defect, months found by $\mathcal{W}$ or SEESAW. A *negative* amount in this column denotes an optimization failure (increased defect, effort, months). Note that such negative results occur only in a small minority of results.

The gray rows indicate any member of a pair that was both statistically significantly different *and* had a lower 50th percentile value. Note that for most pairs, the results are not statistically significantly different (Mann-Whitney, 95% confidence level).

Before commenting on SEESAW vs $\mathcal{W}$, we first note that our results should encourage more use of quality optimization. Observe that, in the majority of cases, *quality optimization works* regardless of how it is implemented (e.g. CBR vs parametric models). In the 52 experiments of Figure **??**, positive quality improvements were seen for $49/52 = 94\%$ experiments (the 3 exceptions are in the defect results of Flight and OSP2).

Another result that should encourage more use of quality optimizers is the reduction in the spreads. In all experiments the amount of uncertainty in the median estimates was reduced. As shown in Figure 13, the reduction in the spread was usually over

	ranges			values	
project	feature	low	high	feature	setting
	prec	1	2	data	3
OSP:	flex	2	5	pvol	2
Orbital	resl	1	3	rely	5
space	team	2	3	pcap	3
plane	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		
	rely	3	5	tool	2
JPL	data	2	3	sced	3
flight	cplx	3	6		
software	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		

	ranges			values	
project	feature	low	high	feature	setting
	prec	3	5	flex	3
OSP2	pmat	4	5	resl	4
	docu	3	4	team	3
	ltex	2	5	time	3
	sced	2	4	stor	3
	KSLOC	75	125	data	4
				pvol	3
				ruse	4
				rely	5
				acap	4
				pcap	3
				pcon	3
				apex	4
				plex	4
				tool	5
				cplx	4
				site	6
	rely	1	4	tool	2
JPL	data	2	3	sced	3
ground	cplx	1	4		
software	time	3	4		
	stor	3	4		
	acap	3	5		
	apex	2	5		
	pcap	3	5		
	plex	1	4		
	ltex	1	4		
	pmat	2	3		
	KSLOC	11	392		

Figure 8: The four NASA case studies. Numeric values *{1, 2, 3, 4, 5, 6}* map to *{very low, low, nominal, high, very high, extra high}*.

Rank	Goal	Change	Median Reduc	Spread Reduc	Reduction Quartiles 50%
1	defects	ReduceFunct	64%	28%	
1	defects	W	54%	32%	
1	defects	Tools&Tech	51%	39%	
1	defects	ProcMaturity	39%	73%	
2	defects	Personel	23%	100%	
3	defects	ReduceQuality	0%	100%	
4	defects	RelaxScedule	-20%	43%	
1	effort	ReduceFunct	62%	28%	
2	effort	W	58%	32%	
2	effort	Tools&Tech	46%	22%	
2	effort	ProcMaturity	24%	76%	
2	effort	ReduceQuality	0%	100%	
2	effort	Personel	0%	105%	
3	effort	RelaxScedule	-13%	35%	
1	months	ReduceFunct	37%	16%	
1	months	Personel	32%	98%	
1	months	W	30%	16%	
1	months	Tools&Tech	29%	26%	
1	months	ProcMaturity	29%	33%	
1	months	ReduceQuality	0%	98%	
2	months	RelaxScedule	-3%	16%	

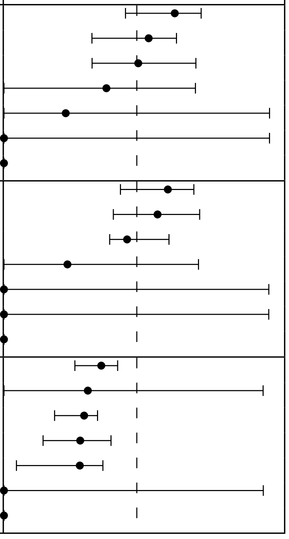


Figure 9: Comparing defect, effort, and month estimation reduction percentages ($100 * \frac{initial-final}{intial}$ of drastic business decisions vs $\mathcal{W}$'s recommendations for the Ground case study.

Rank	Goal	Change	Median Reduc	Spread Reduc	Reduction Quartiles 50%
1	defects	ReduceFunct	64%	28%	
1	defects	W	54%	32%	
1	defects	Tools&Tech	51%	39%	
1	defects	ProcMaturity	39%	73%	
1	defects	Personel	23%	100%	
1	defects	ReduceQuality	0%	100%	
2	defects	RelaxScedule	-20%	43%	
1	effort	ReduceFunct	62%	28%	
1	effort	W	58%	32%	
1	effort	Tools&Tech	46%	22%	
1	effort	ProcMaturity	24%	76%	
1	effort	ReduceQuality	0%	100%	
1	effort	Personel	0%	105%	
2	effort	RelaxScedule	-13%	35%	
1	months	ReduceFunct	37%	16%	
1	months	Personel	32%	98%	
1	months	W	30%	16%	
1	months	Tools&Tech	29%	26%	
1	months	ProcMaturity	29%	33%	
1	months	ReduceQuality	0%	98%	
2	months	RelaxScedule	-3%	16%	

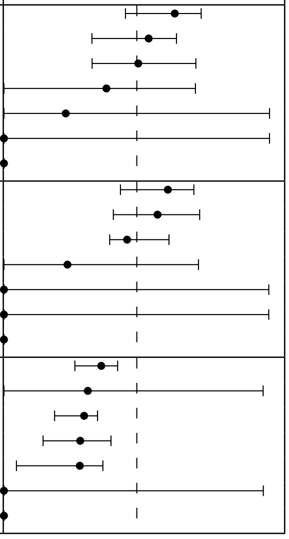


Figure 10: Comparing defect, effort, and month estimation reduction percentages ($100 * \frac{initial-final}{intial}$ of drastic business decisions vs $\mathcal{W}$'s recommendations for the Flight case study.

61%. This is a major advantage of quality optimizers since uncertainty is an serious issue that plagues the managers of software engineering projects.

The spread reductions were larger than the median reductions. As shown in Figure 13, the expected median reduction in any quality estimate was only 15%. 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 quality optimizers finding recommendations that resulted in an order of magnitude reduction in effort *and* defects *and* development

Rank	Goal	Change	Median Reduc	Spread Reduc	Reduction Quartiles 50%
1	defects	W	61%	31%	
2	defects	ProcMaturity	51%	26%	
2	defects	ReduceFunct	46%	34%	
2	defects	Tools&Tech	39%	32%	
2	defects	ReduceQuality	0%	382%	
2	defects	Personel	0%	100%	
3	defects	RelaxScedule	-30%	78%	
1	effort	W	60%	28%	
2	effort	ProcMaturity	51%	29%	
2	effort	ReduceFunct	48%	36%	
2	effort	Tools&Tech	47%	45%	
2	effort	ReduceQuality	5%	257%	
2	effort	Personel	0%	100%	
3	effort	RelaxScedule	-21%	64%	
1	months	ProcMaturity	31%	15%	
2	months	W	30%	17%	
2	months	Personel	29%	98%	
2	months	Tools&Tech	25%	17%	
2	months	ReduceFunct	25%	9%	
2	months	ReduceQuality	4%	61%	
3	months	RelaxScedule	-7%	16%	

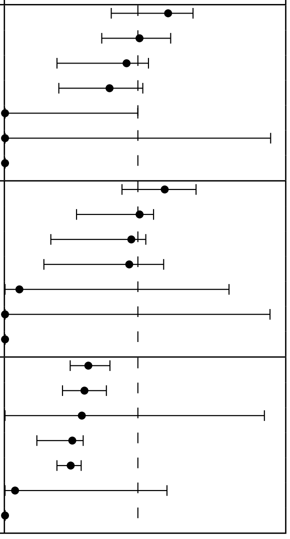


Figure 11: Comparing defect, effort, and month estimation reduction percentages ($100 * \frac{initial-final}{intial}$ of drastic business decisions vs $\mathcal{W}$'s recommendations for the OSP case study.

Rank	Goal	Change	Median Reduc	Spread Reduc	Reduction Quartiles 50%
1	defects	W	64%	40%	
1	defects	Tools&Tech	57%	24%	
1	defects	ReduceFunct	50%	29%	
1	defects	Personel	28%	75%	
1	defects	ProcMaturity	24%	38%	
1	defects	ReduceQuality	0%	163%	
1	defects	RelaxScedule	-17%	71%	
1	effort	ReduceFunct	63%	27%	
1	effort	W	60%	45%	
1	effort	Toolss&Tech	57%	36%	
1	effort	ReduceQuality	50%	100%	
1	effort	Personel	19%	78%	
1	effort	ProcMaturity	14%	49%	
2	effort	RelaxScedule	-43%	91%	
1	months	W	35%	21%	
1	months	Toolss&Tech	30%	15%	
1	months	ReduceFunct	26%	12%	
1	months	Personel	25%	45%	
1	months	ProcMaturity	12%	21%	
1	months	ReduceQuality	6%	98%	
2	months	RelaxScedule	-16%	25%	

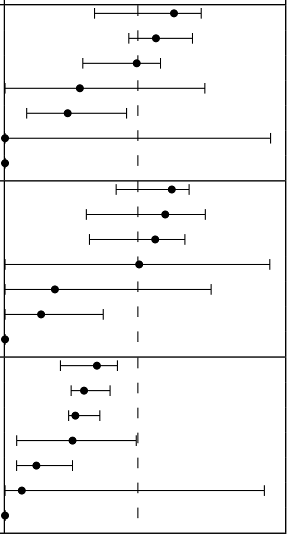


Figure 12: Comparing defect, effort, and month estimation reduction percentages ($100 * \frac{initial-final}{intial}$ of drastic business decisions vs $\mathcal{W}$'s recommendations for the OSP2 case study.

time. That would suggest that the managers of software engineering projects are routinely missing changes that would significantly improve their projects.

Another feature to note is that, with only a few exceptions, the median optimizations obtained from case-based reasoning or parametric modeling are very similar. For example, in median MONTHS results (top right of Figure **??**), within each pair of treatments, the change in the median months values is very similar:

- 19 vs 18% change (for Ground)
- 14 vs 9% change (for Flight)
- 11 vs 12% change (for OSP)
- 12 vs 14% change (for OSP2)

That is, projects can contain an an inherent set of constraints that cannot be changed, even by smart algorithms. Certainly, we can fine tune the structure of a project to obtain some improvements in effort, defects, and months but managers should not expect a magic silver bullet that offers orders of magnitude improvement in their software process.

Turning now to the main point of this paper, we conducted statistical tests on each pair of $\mathcal{W}$ 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 15. Note that, in the majority case ($\frac{18}{24}$), $\mathcal{W}$'s case-based reasoning performs as well as $SEESAW's$ parametric modeling.

Also, when the performance results were different, case-based reasoning did better than parametric modeling ($\frac{6}{24}$)- sometimes spectacularly so, Observe the median DEFECT OSP2 results (last line, top row, middle of Figure **??**): SEESAW's recommendations resulted in a dramatic increase in the number of delivered defects (2612 to 7797). This result shows that $\mathcal{W}$'s modest decrease in defects (13%) is actually far better than those found by the other approach.

In summary, the simple case-based reasoning of $\mathcal{W}$ performs just as well, or better, than SEESAW's elaborate parametric modeling.

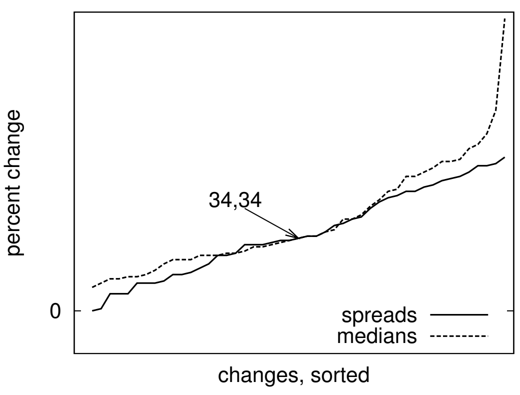
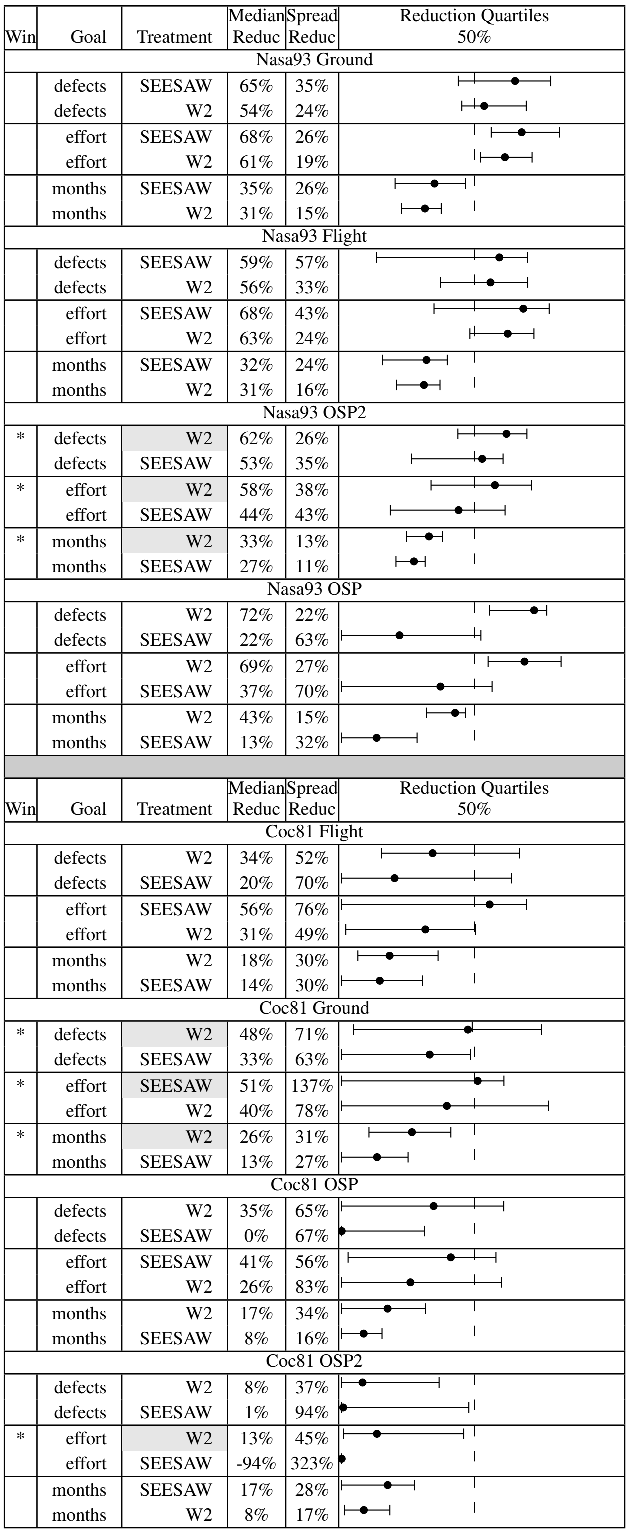


Figure 13: Range of changes in median and spread generated by applying the recommendations of either $\mathcal{W}$ or SEESAW. The median observed changes were (34, 34)% for (medians, spreads), respectively. For the sake of brevity, this graph ignores the -94% outlier value seen in OSP2 defects for SEESAW.

6.2 $\mathcal{W}$ Performance Across Multiple Datasets

Because $\mathcal{W}$ makes no underlying model assumptions, we aren't limited to USC COCOMO for our evaluations. To demonstrate the effectiveness of $\mathcal{W}$ in any data environment, we offer median reductions for effort reduction for five arbitrary datasets from `http://promised`
The model-agnostic simpilicity of $\mathcal{W}$ made implementing these tests easy as one need only describe a query space and a target utility measure. In the case of these five datasets, software effort was the common target for reduction.

Given that we did not have access to case studies as we did with NASA93 and COC81 (ground, flight, osp, and osp2) for these datasets, synthetic queries were developed. Three queries were generated for each of the five datasets. The first contained the entire



Win	Goal	Treatment	Median Reduc	Spread Reduc	Reduction Quartiles 50%
		Nasa93 Ground			
	defects	SEESAW	65%	35%	
	defects	W2	54%	24%	
	effort	SEESAW	68%	26%	
	effort	W2	61%	19%	
	months	SEESAW	35%	26%	
	months	W2	31%	15%	
		Nasa93 Flight			
	defects	SEESAW	59%	57%	
	defects	W2	56%	33%	
	effort	SEESAW	68%	43%	
	effort	W2	63%	24%	
	months	SEESAW	32%	24%	
	months	W2	31%	16%	
		Nasa93 OSP2			
*	defects	W2	62%	26%	
	defects	SEESAW	53%	35%	
*	effort	W2	58%	38%	
	effort	SEESAW	44%	43%	
*	months	W2	33%	13%	
	months	SEESAW	27%	11%	
		Nasa93 OSP			
	defects	W2	72%	22%	
	defects	SEESAW	22%	63%	
	effort	W2	69%	27%	
	effort	SEESAW	37%	70%	
	months	W2	43%	15%	
	months	SEESAW	13%	32%	

Win	Goal	Treatment	Median Reduc	Spread Reduc	Reduction Quartiles 50%
		Coc81 Flight			
	defects	W2	34%	52%	
	defects	SEESAW	20%	70%	
	effort	SEESAW	56%	76%	
	effort	W2	31%	49%	
	months	W2	18%	30%	
	months	SEESAW	14%	30%	
		Coc81 Ground			
*	defects	W2	48%	71%	
	defects	SEESAW	33%	63%	
*	effort	SEESAW	51%	137%	
	effort	W2	40%	78%	
*	months	W2	26%	31%	
	months	SEESAW	13%	27%	
		Coc81 OSP			
	defects	W2	35%	65%	
	defects	SEESAW	0%	67%	
	effort	SEESAW	41%	56%	
	effort	W2	26%	83%	
	months	W2	17%	34%	
	months	SEESAW	8%	16%	
		Coc81 OSP2			
	defects	W2	8%	37%	
	defects	SEESAW	1%	94%	
*	effort	W2	13%	45%	
	effort	SEESAW	-94%	323%	
	months	SEESAW	17%	28%	
	months	W2	8%	17%	

Figure 14: Range of changes in median and spread generated by applying the recommendations of either $\mathcal{W}$ or SEESAW.

space of possible project attribute values (All), 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 attribute (Proj1, Proj2). These queries represent more common restrictions on possible changes for a given software project.

Algorithm	Wins	Losses	Ties
W	6	1	17
SEESAW	1	6	17

Figure 15: Win/Loss/Tie table for statistically significant reductions across all goals with the Nasa Flight, Ground, OSP, and OSP2 projects for both the Nasa93 and Coc81 datasets.

Effort reductions can be seen in figures 17 and 16. The chart in figure 17 shows strong improvement in median effort for the Telecom and Miyazaki datasets, with strong performance in spread reduction across all datasets. While the Finnish, China, and Kemerer datasets show only marginal or no improvement in median effort, the certainty of their estimations is improved via a reduction in spread.

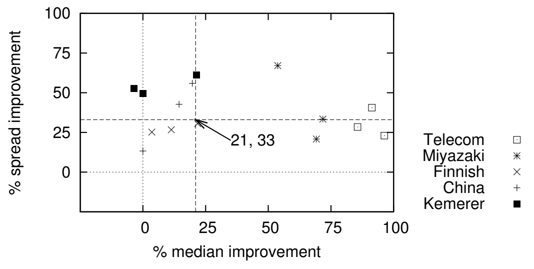


Figure 16: Plot showing the distribution of median and spread reductions in software effort for five unique datasets.

		Improvement	
dataset	query q	median	spread
Telecom	Proj1	96%	23%
Telecom	Proj2	91%	41%
Telecom	All	86%	28%
Miyazaki	All	78%	33%
Miyazaki	Proj2	69%	21%
Miyazaki	Proj1	53%	67%
Finnish	All	22%	31%
Finnish	Proj2	11%	27%
Finnish	Proj1	4%	25%
China	All	20%	55%
China	Proj2	14%	43%
China	Proj1	0%	13%
Kemerer	Proj1	21%	61%
Kemerer	Proj2	0%	49%
Kemerer	All	-4%	53%
	median	21%	33%

Figure 17: Effort estimation improvements $(100 * \frac{initial - final}{intial})$ **for five unique datasets. Sorted by median improvement. Gray cells represent no improvement in effort estimates.**

7. DISCUSSION

7.1 Search-based Software Engineering

Previously [10], we have explored the connection of SEESAW to search-based SE (SBSE) [11]. In summary, 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. SBSE has been applied to many problems in software engineering (e.g., requirements engineering [12]) but most often in the field of software testing [2]. Harman's writing inspired us to try simulated annealing (SA) to search the what-ifs in untuned COCOMO models [18]. For quality optimization, however, we found that search methods taken from the constraint satisfaction literature out-perform SA [10].

7.2 Model-lite

We said above that CBR was *model-lite*, but not *model-free*. We hesitate to call CBR *model-free*, lest we incur the wrath of Janet Kolodner or Roger Shank [24]. Kolodner and Shank regard CBR as a *model* of human cognition where knowledge in a context-dependent manner, according to the task at hand. This construct may differ from context to context but the search mechanisms by which the construct is built (CBR) is constant.

To expand on that point, we note that "model" has at least two definitions:

1. A hypothetical description of a complex entity or process.
2. A plan to create, according to a model or models.

The first definition is closest to Shepperd's definition of "model-based systems". According to Shepperd [25] software effort estimation methods separate into "human-centric" techniques and "model-based" techniques. In the former, humans produce their recommendations without using some externalizable representation. In the latter, a variety of techniques may be used which, according to Shepperd, divide into algorithmic/parametric models (like COCOMO) and induced prediction systems (which include regression, rule induction, CBR, and many others).

We can marry Shepperd's view with that of Kolodner and Shank by specializing the definition of model-based systems. Extending Shepperd's ontology, we say that model-based systems can be sorted according to how much modeling they assume prior to induction. At one end of that sort order, we have parametric models like COCOMO. We call these *model-heavy* since they conform to the first definition of "model", shown above. At the other end of that sort are the *model-lite* methods like CBR. These model-lite methods conform to the second definition of "model". Note that this second definition is closest to Kolodner and Shank's view on CBR; i.e. the CBR model is a recipe for generating context-dependent knowledge.

8. CONCLUSION

Advocates of reconstructive memory such as Barlett [4], Kolodner [13], or Shank [24] argue that *we make it up as we go along*. In case-based reasoning (CBR), inference repeats every time there is a new query. Our reading of the papers at this conference is that, except for a few papers that deal with reasoning-by-analogy (e.g. [3]), most of this community avoids the model-lite approach of CBR.

Proponents of parametric models argue that there exist *domain-independent models* which can be *tuned* to local details. In this approach, reasoning can take the form of a data miner learning values for tune-able attributes of a parametric model like Equation 1. In this way, learning can happen once and users can use the tuned model for all future queries.

Unfortunately, these supposedly domain-independent models (like COCOMO) suffer from massive internal variance (see Figure 1). Previously, we have tried to manage internal variance of this problem with SEESAW: an AI algorithm that sought stable conclusions across the space of possible tunings within a parametric model. While a successful prototype, SEESAW has disadvantages:

- Dependency on a particular parametric model
- A requirement that all the data be in a format acceptable to that model
- Too many arbitrary internal design decisions
- Slow runtimes
- A code base that proved too large to maintain, modify, and add support for more models

With a result supporting CBR, this paper finds little to recommend from SEESAW over the $\mathcal{W}$ case-based reasoning tool. Standard CBR applies a query q to find relevant examples from a set of cases C using the retrieve-reuse-revise-retain loop of Figure 2. $\mathcal{W}$ extends standard CBR by learning an adaption of q, called q', that retrieves *better* quality examples. Based on the analysis of [8] and this paper, we recommend $\mathcal{W}$ on several grounds:

- $\mathcal{W}$ finds similar, or better, results than SEESAW (see Figure 15).
- $\mathcal{W}$ is simpler to code: 200 lines of AWK as opposed to the 5000 lines of LISP code used in SEESAW.
- $\mathcal{W}$ is faster to run: the above experiments took minutes for $\mathcal{W}$, but hours for SEESAW.
- $\mathcal{W}$ is simpler to maintain since, in CBR, "maintenance" means nothing more than "add more cases".
- $\mathcal{W}$ makes no use of an underlying model and is therefore free from the assumptions of parametric modeling. Hence it can be applied to more data sets. For example, SEESAW requires data to be in the COCOMO format but $\mathcal{W}$ has been applied to numerous data sets in other formats [8].

Having said that, there is one situation where we'd recommend SEESAW over $\mathcal{W}$. Like all CBR systems, $\mathcal{W}$ needs cases. If there is *no* local data, then SEESAW would be the preferred (only) option.

What then should we say about the premise of PROMISE; i.e. that "modeling" is an appropriate method for understanding SE projects? Our answer is two-fold. Firstly, there is insufficient evidence in this paper to make the conclusion that CBR *always* beats model-heavy methods like parametric models. Nevertheless, these results clearly motivate further exploration and comparison between the value of CBR and model-heavy techniques. For example, at our lab we are exploring very fast clustering methods to support scaling CBR to very large data sets.

Secondly, there are at least two kinds of "models." In the traditional model-heavy definition, models are specific *products* that can be applied to multiple domains. In the CBR model-lite definition, a model is a *process* that generates many products, each of which is customized to the particulars of a local domain. In this paper and [19] we have seen the following advantages of CBR: easy implementation, fast runtimes, easy maintenance, able to be applied to more data, and out-performance of model-heavy methods. If these advantages apply in other problem domains, we speculate that the future of PROMISE will be "models-as-process" and not "models-as-products".

Acknowledgments

This work was partially funded by the United Stated National Science Foundation (CCF-1017263).

9. REFERENCES

[1] A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intellegence Communications*, 7:39–59, 1994.

[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] M. Azzeh, D. Neagu, and P. Cowling. Improving analogy software effort estimation using fuzzy feature subset selection algorithm. In *PROMISE '08: Proceedings of the 4th international workshop on Predictor models in software engineering*, pages 71–78, 2008.

[4] F. Bartlett. *Remembering: A study in experimental and social psychology*. The Cambridge University Press, 1932.

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

[6] B. Boehm. Safe and simple software cost analysis. *IEEE Software*, pages 14–17, September/October 2000. Available from `http://www.computer.org/certification/beta/Boehm_Safe.pdf`.

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

[8] A. Brady, T. Menzies, J. Keung, O. El-Rawas, and E. Kocaguneli. Case-based reasoning for reducing software development effort. *Journal of Software Engineering and Applications*, 2010.

[9] N. Fenton, M. Neil, W. Marsh, P. Hearty, L. Radlinski, and P. Krause. Project data incorporating qualitative factors for improved software defect prediction. In *PROMISE'09*, 2007. Available from `http://promisedata.org/pdf/mpls2007FentonNeilMarshHeartyRadlinskiKrause.pdf`.

[10] 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`.

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

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

[13] J. Kolodner. Reconstructive memory: A computer model. *Cognitive Science*, 7(4):281–328, 1983.

[14] E. Loftus. Our changeable memories: legal and practical implications. *Nature Rev. Neurosci.*, pages 231–234, 2003.

[15] T. Menzies, Z. Chen, D. Port, and J. Hihn. Simple software cost estimation: Safe or unsafe? In *Proceedings, PROMISE workshop, ICSE 2005*, 2005. Available from `http://menzies.us/pdf/05safewhen.pdf`.

[16] 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`.

[17] T. Menzies, O. Elrawas, B. Barry, R. Madachy, J. Hihn, D. Baker, and K. Lum. Accurate estimates without calibration. In *International Conference on Software Process*, 2008. Available from `http://menzies.us/pdf/08icsp.pdf`.

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

[19] T. Menzies and J. Kiper. How to argue less, 2001. Available from `http://menzies.us/pdf/01jane.pdf`.

[20] 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`.

[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*, 14:213–225, July 2009. Available from `http://menzies.us/pdf/09nodata.pdf`.

[22] 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`.

[23] R. C. Schank. *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*. Cambridge University Press, New York, NY, USA, 1983.

[24] R. C. Schank and R. P. Abelson. *Scripts, plans, goals and understanding: an inquiry into human knowledge structures*. Erlbaum, 1977.

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

[26] M. Shepperd and C. Schofield. Estimating software project effort using analogies. *IEEE Transactions on Software Engineering*, 23(12), November 1997. Available from `http://www.utdallas.edu/~rbanker/SE_XII.pdf`.

[27] I. Watson. *Applying case-based reasoning: techniques for enterprise systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

APPENDIX

This appendix lists the minimum and maximum m values used for Equation 5 and Equation 6. In the following, m_α and m_β denote COCOMO's linear and exponential influences on effort/cost, and m_γ and m_δ denote COQUALMO's linear and exponential influences on number of defects.

Their are two sets of effort/cost multipliers:

1. 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.
2. 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.

Their m ranges, as seen in 161 projects [6], are:

$$\left(0.073 \leq m^+_\alpha \leq 0.21\right) \wedge \left(-0.178 \leq m^-_\alpha \leq -0.078\right) \quad (8)$$

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

$$-1.56 \leq m_\beta \leq -1.014 \tag{9}$$

Similarly, there are two sets of defect multipliers and scale factors:

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

COQUALMO divides into three models describing how defects change in requirements, design, and coding. These tunings options have the range:

$$\begin{gathered}
requirements \begin{cases} 0 \leq m_\gamma^+ \leq 0.112 \\ -0.183 \leq m_\gamma^- \leq -0.035 \end{cases} \\
design \begin{cases} 0 \leq m_\gamma^+ \leq 0.14 \\ -0.208 \leq m_\gamma^- \leq -0.048 \end{cases} \\
coding \begin{cases} 0 \leq m_\gamma^+ \leq 0.14 \\ -0.19 \leq m_\gamma^- \leq -0.053 \end{cases}
\end{gathered} \tag{10}$$

The tuning options for the defect removal features are:

$$\begin{array}{rl}
\forall x \in \{1..6\} & SF_i = m_\delta (x-1) \\
requirements: & 0.08 \leq m_\delta \leq 0.14 \\
design: & 0.1 \leq m_\delta \leq 0.156 \\
coding: & 0.11 \leq m_\delta \leq 0.176
\end{array} \tag{11}$$

where m_δ denotes the effect of i on defect removal.