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

ABSTRACT

Machine learning has been incredibly successful. We are drowning in choice as to how we can build software *quality optimizers*, programs that find project options that change quality measures like defects, development effort (total staff hours), and time (elapsed calendar months). This paper presents a general strategy and specific implementation for *contrast set learning*, a means of determining what separates two distinct populations. We expand upon previous work with $\mathcal{W}$, a simple, calibrationless, model-agnostic, case-based reasoning algorithm that implements contrast-set learning.

The optimized $\mathcal{W}2$ algorithm performs as well as our previous $\mathcal{W}$ algorithm, runs in linear time, and is easier to explain. We show a worked example where $\mathcal{W}2$ is applied to a hypothetical instance of Brooks' Law, where a project has recently brought onboard a large number of inexperienced programmers.

We present an extended certification of $\mathcal{W}2$'s performance across multiple, arbitrary datasets as well a practical example of $\mathcal{W}2$'s performance against drastic management decisions. We demonstrate that $\mathcal{W}2$ performs just as well or better for reducing software cost across multiple goals such as development effort, project defects, and completion time.

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

PROMISE '10 Timisoara, Romania

1. INTRODUCTION

Imitation *is* the sincerest form of flattery. As humans we tend to emulate our superiors in order to learn from them. For example, a writer may read their favorite author for inspiration, an artist may draw influence from the the works of the Renaissance, and a programmer may subscribe to the blog of a famous startup-turned-empire. From them we can gleam what separates the best of us from the rest of us, so that we might repeat their successes.

Our drive to imitate translates not only to personal traits, but software projects as well. Certainly a software project manager also seeks to imitate past sucesses and avoid prior failures. However, just as the artist studies another artist in a similar medium, the manager seeks guidance from projects in a similar ecosystem. One could generalize that from these similar projects, a general strategy is to examine the *best* performers, and adopt the attributes that are most unique to the *best* rather than the *rest*. We call this strategy *contrast-set learning*.

But how should we use this apriori knowledge to infer about new instances? Should we extrapolate from old data to build a parametric model; e.g. using a Bayes net [**?**], or the linear equations of COCOMO [**?**, **?**]? Or is it best to reason directly from data, without an intervening parametric model, using case-based reasoning (CBR) [**?**]?

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.

Prior work [**?**] compared 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 [**?**] and has been applied to numerous domains [**?**, **?**, **?**, **?**, **?**, **?**]. $\mathcal{W}$ was first introduced in [**?**] but

that report includes no comparisons with other quality optimizers. More recent work with W presented a simple quality optimization algorithm and compared it to SEESAW [**?**]. From our comparisons between SEESAW and $\mathcal{W}$ in [**?**] we demonstrated:

- $\mathcal{W}$ 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 [**?**].

In this paper we extend our work with $\mathcal{W}$ and SEESAW to further the discussion on model-based vs case-based reasoning methodologies. We relate $\mathcal{W}$ and SEESAW's implementation to the general algorithm of contrast-set learning, and how this accessible approach is able to extract distinguishing attributes between two populations.

We extend $\mathcal{W}$ into $\mathcal{W}2$, a optimization of the $\mathcal{W}$ algorithm. Our discussion of $\mathcal{W}2$ offers the the following improvements over prior work:

- *Optimization* - $\mathcal{W}2$ removes $\mathcal{W}$'s $O(n^2)$ nearest neighbor calculation and replaces it with a simple linear-time filter of the historical data. Despite this simplification, $\mathcal{W}2$ performs as well as $\mathcal{W}$ (Figure **??**).
- *More Extensive Certification* - Prior work benchmarked the model-lite approach of $\mathcal{W}$ with model-dependent approach of SEESAW. A direct comparision between $\mathcal{W}$ and SEESAW required the exclusive use of the COCOMO model format. In this paper we have extended $\mathcal{W}2$'s performance evaluation into new, arbitrary datasets in order to gauge $\mathcal{W}2$'s usefulness in a wide variety of instutitions and applications.
- *Better Explanation* - $\mathcal{W}2$ is even simpler to explain than $\mathcal{W}$. We have provided a visual example of $\mathcal{W}2$ that demonstrates the three main steps of $\mathcal{W}2$: Relevancy Filtering, Contrast Set Learning, and Holdout Set Performance Estimation.
- *Direct Application* - When a software project veers away from perceived estimates, drastic changes may be needed to get it back on track. We compare the impact of several drastic management decisions, such as reducing functionality to cut development effort, to alternative decisions learned with $\mathcal{W}2$.

We conclude that, for the task of quality optimization, $\mathcal{W}2$ represents an improvement over $\mathcal{W}$(Figure **??**). We also recommend $\mathcal{W}2$'s case-based reasoning over SEESAW's parametric modeling (Figure **??**). For instances of drastic software changes, $\mathcal{W}2$ offers alternatives that can be equally effective in improving software quality (Figures **??**, **??**, **??**, **??**). Also, across multiple datasets, $\mathcal{W}2$'s contrast-set learning demonstrates improvements in software quality regardless of the underlying data description (Figure **??**).

While we offer no conclusion on the general merits of case-based reasoning compared to parametric modeling, we offer that $\mathcal{W}2$, contrast-set learning, and case-based reasoning are important tools for software quality optimization.

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 [**?**]):

$$Effort = a * Loc^b * \prod_i \beta_i x_i \quad (1)$$

where x_i are one of the *effort multipliers* shown in Figure **??** (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 [**?**, **?**], Boehm offers values to (a, b, β_i) to three significant figures. Previously [**?**], we have reported that such precision is somewhat optimistic since β_i has a very large variance. The plot at the bottom of Figure **??** 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 **??** jump from negative to positive.

We have seen instability in other datasets, including the COC81 data used by Boehm to derive Equation **??** [**?**]. 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 **??**) 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

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

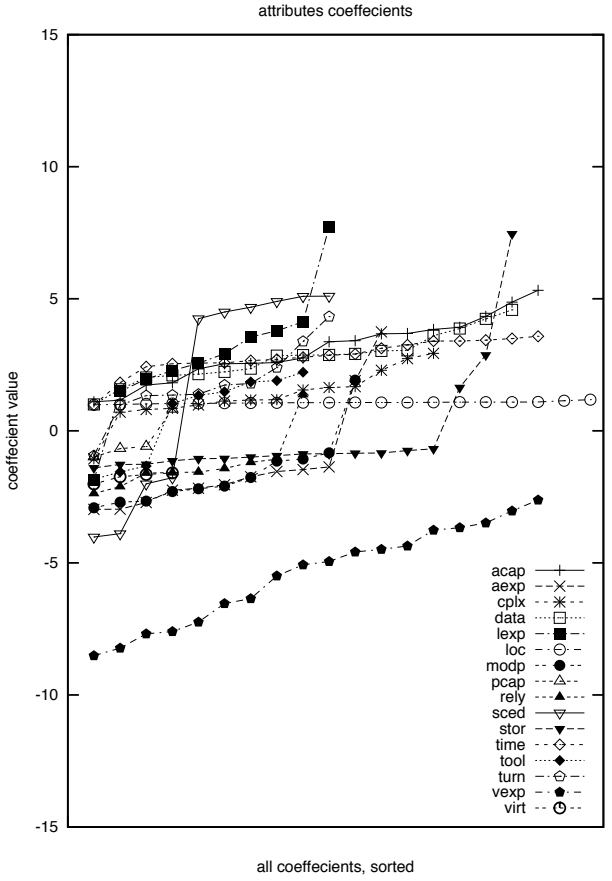

Figure 1: COCOMO 1 effort multipliers, and the sorted coefficients found by linear regression from twenty 66% sub-samples (selected at random) from the NASA93 PROMISE data set; from [?]. Prior to learning, training data was *linearized* in the manner recommended by Boehm (x was changed to $log(x)$; for details, see [?]). 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.

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 [?].*

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 [**?**].

In AI research, Janet Kolodner [**?**] 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 [**?**] 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.

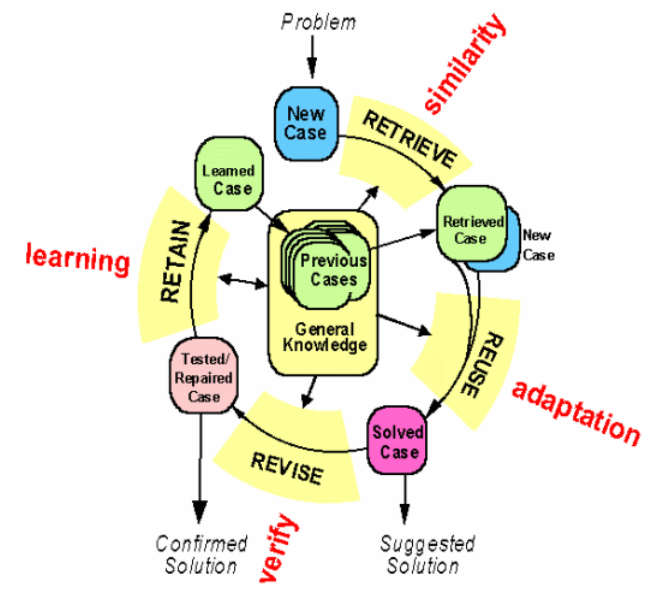

Figure 2: Four steps of CBR, from `http://www.peerscience.com/intro_cbr.htm`.

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 [**?**].

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 the general principle of contrast set learning behind quality optimization, then describes two specific implementations using SEESAW's parametric models or $\mathcal{W}$'s case-based reasoning.

3.1 Contrast Set Learning (CSL)

One process for self-improvement is to emulate those around you that are doing well. For example, imagine a failing student seeking recommendations to improve their grades. Standard parental advice may be to simply study more. However, while such general platitudes may indeed bring improvement, they ignore any local lessons about their life that may bring more success with less effort.

Instead, students being social creatures, they seek out advice from those around them. Given that close friends and colleagues are most likely under the same pressures, it makes sense to seek advice from those in similar circumstances. Then, a rationally-minded student may divide their friends and colleages into two groups: those doing well (to serve as role models), and those not doing so well (to serve as cautionary tales). Finally, the student adopts as many traits they perceive as unique to the role models.

Such a processes allows for multiple, targeted avenues of improvement compared to generic idioms such as "study more." So, the student finds that by avoiding Tuesday parties, asking questions after class, and sitting towards the front of the room, success is achievable. In other words, local lessons offer a more tailored approach to improvement.

Contrast set learning (CLS) applies this process by asking the question "What are my role models doing that I'm not?" Formally, this takes place in three steps:

- *Relevancy Filtering* - Find examples similar to the problem at hand.
- *Utility Separation* - Divide the relevant examples into two populations based on some utility measure: those I want to imitate (the *best*) and those I don't want to imitate (the *rest*).
- *Contrast Set Generation* - Perform a greedy search on attributes that occur more often in *best* than in *rest*. Rank these attributes by some *score* that favors contrast, biasing towards attributes that occur often in *best* but rarely or never in *rest*.

A simple strategy to score more favorably towards attributes that occur most often in the best case is to square the number of times they occurs. 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 [**?**], the ensuring rare attributes are weighted appropriately:

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

From this measure we need only sort each attribute by it's $like$ score to prioritize our recommendations. Thus, we establish a means for finding attributes that most drive us towards our desired goal. An alternative to Equation **??** is to log the odds ratio between an attribute appearing in *best* rather than *rest* [**?**].

```
@project brookslaw
@attribute apex 2
@attribute plex 1 2
@attribute ltex 1 2 3
@attribute ?pmat 2 3
@attribute ?rely 3 4 5
@attribute ?data 2 3
@attribute ?cplx 4 5
@attribute ?time 4 5
@attribute ?stor 3 4 5
@attribute ?pvol 2 3 4
@attribute ?acap 3 4 5
@attribute ?pcap 3 4 5
@attribute ?tool 3 4
@attribute ?sced 2 3
```

Figure 3: The Brooks' Law Query for the NASA93 dataset in COCOMO II format.

3.2 Implementing CSL with $\mathcal{W}2$

CSL describes a general strategy for reasoning about two distinct populations. Because CSL requires no underlying model to implement, we originally created $\mathcal{W}$ to add CSL decision power to case-based reasoning software cost estimates. Upon further experimentation, we improved upon $\mathcal{W}$ by removing the kth nearest neighbor calculation in favor of simply using our *overlap* measure to perform relevancy filtering. The original description of $\mathcal{W}$ can be found in [**?**]. We offer a statement on performance between $\mathcal{W}$ and $\mathcal{W}2$'s in the results section and Figure **??**.

$\mathcal{W}2$ answers the question: "What can I change about this project to make it more like best cases?" In other words, "How can I best imitate what I aspire to be?" To answer this, $\mathcal{W}2$ requires two sets of information:

- A set of historical cases C_i with quantified attributes (say, management experience, lines of code) and some measure of utility (say, effort in man-months, total defects, months for development). All attributes have been discretized into a small number of ranges (e.g. manager experience $\in \{1, 2, 3, 4, 5\}$ denoting very low, low, nominal, high, very high respectively)
- A query q describing the current project seeking improvement, with defined ranges for potential changes, as well as any constraints that cannot be changed. 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 **??**.

$\mathcal{W}2$ is easily demonstrated visually. Figure **??** demonstrates a query representing a project query q involving Brooks' Law [**?**] using 93 NASA project cases in COCOMO format. In the 1970's, Brooks noted that software production is a very human-centric activity and managers need to be aware of the human factors that increase/decrease productivity. For example, a common practice at that time at IBM was to solve deadline problems by allocating more resources. In the case of programming, this meant adding more programmers to the team. Brooks argued that this was an inappropriate response since, according to Brooks' law "adding manpower [sic] to a late software project makes it late". The reason for this slowdown is two-fold:

- The more people involved the greater the communication overhead. While this is certainly an issue if all parts of the

software system are accessible to all other parts, with an intelligent module design, this first issue can be mitigated.

- The second issue is more fundamental. Software construction is a complex activity. Newcomers to a project suffer from inexperience in the tools, the platform, the problem domain, etc.

The query in Figure **??** models this second issue. Attributes with a ? represent controllable attributes, with *apex, plex*, and *ltex* representing the uncontrollably lower ratings of analyst experience, programmer language experience, and language and tool experience, respectively.

First, cases are randomly separated into 67% Training and 33% Testing sets. Then, $\mathcal{W}2$ implements the same three steps used for CSL. Finally, $\mathcal{W}2$ estimates the impact of its recommendations:

3.2.1 Relevancy Filtering

Relevancy Filtering: Brooks' Law Query, NASA93 Dataset																
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
(39 cases omitted)																
54	4	4	4	4	5	4	5	6	6	3	4	4	3	3	8211	7
52	4	4	4	4	5	4	5	6	6	3	4	4	3	3	1645.9	7
51	4	4	4	4	5	4	5	6	6	3	4	4	3	3	4178.2	7

Figure 4: Excerpt of the NASA93 dataset demonstrating overlap between the query and historical cases.

From Training, 20 cases are selected with the highest total *overlap* with the project query (Figure **??**). For example, if a case had a schedule rating of *high*, and q defines the controllable scedule range as potentially *high* or *very high*, then that attribute is said to *overlap* with the query. This is the *retrieve* step in Standard CBR nomencalature.

3.2.2 Utility Separation

The 20 cases are then sorted by some utility measurement, with the top 5 cases placed into the *best* set and the remaining 15 into the *rest* set (Figure **??**). For datasets with multiple goals, such as the NASA93 and COC81 datasets that contain project effort, defects, and months, a utility function normalizes each value into a single utility "score". Other datasets simply minimize software effort in man-months. This is the first half of the CBR *reuse* (or adapt) step.

BEST Set														
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

REST Set															
row	*apex*	*plex*	*ltex*	pmat	rely	data	cplx	time	stor	pvol	acap	pcap	tool	sced	effort
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 5: The "Best" and "Rest" sets. Top 5 cases with the most desirable effort (lowest) are in Best while the remaining 15 are in Rest.

3.2.3 Contrast Set Generation

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

Figure 6: Contrast between the "Best" and "Rest" sets. Contrast values computed by $like = \frac{freq(x|best)^2}{freq(x|best)+freq(x|rest)}$ rank each attribute value according to contrast. Higher $like$ values for an attribute imply its association with smaller effort values.

Changes to q are ranked according to equation **??**. This sorted order S defines a set of candidate q' queries that use the first *i-th*

entries in S (Figure **??**):

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

In the Brooks' Law example, $\mathcal{W}2$ learns that *pmat=3* scores the highest for reducing development effort. This is the last half CBR *reuse* (or adapt) step.

At this point, $\mathcal{W}2$ has created a list of recommendations that best drive q towards more desirable utility measures (Figure **??**). However, we do not yet have an estimate as to the impact of applying these recommendations. The next phase of $\mathcal{W}2$ estimates the improvement in software quality after applying q'_i.

3.2.4 Estimating Impact

Brooks' Law Query on Test Set															
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
21	3	3	4	4	4	2	4	3	3	2	3	3	3	2	60	9
10	5	3	4	3	3	2	4	3	3	2	4	5	3	3	72	10
71	4	4	4	2	3	2	4	3	5	2	4	4	3	2	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
31	4	2	4	4	3	5	4	5	5	2	5	3	3	2	170	10
73	4	4	4	2	3	3	4	3	5	2	4	4	3	2	300	10
45	4	3	4	3	4	4	3	3	3	2	3	4	3	3	400	8
79	3	4	4	4	4	3	5	4	4	2	4	3	3	3	409	10
84	5	1	1	4	4	2	5	5	6	2	5	5	4	3	430	11
67	4	3	4	3	5	3	4	4	3	2	4	4	3	3	444	11
80	3	4	4	4	4	3	5	4	4	2	4	3	3	3	703	10
60	3	4	4	3	3	2	4	3	3	2	5	5	3	3	720	10
76	4	4	4	2	4	5	4	3	5	2	4	4	3	2	756	9
46	4	3	4	2	2	3	3	3	3	2	4	5	3	3	2400	8
59	5	1	4	2	5	2	6	6	5	2	4	4	3	3	4560	10

Figure 7: Applying $q \cup S_1$ to the 20 most relevant cases from the testing set. Here, cases highlighted represent those that include $pmat = 3$, learned during training.

Brooks' Law Query $\cup$ *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

Figure 8: The testing set with all cases not containing $pmat = 3$ removed. The impact of applying $pmat = 3$ is reported as the median effort value of the cases. In this case, 81.

Query	Effort Median	Effort Spread	Effort Distrubtion 366
q (Initial)	235	508	
$q \cup pmat = 3$ (Final)	81	352	

Figure 9: Result of applying the learned constraint $pmat = 3$ to the Brooks' Law query q during testing. The median estimate reduction from 235 to 81 represents a 66% reduction is software effort by applying $pmat = 3$.

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

Figure 10: $\mathcal{W}$'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)

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

On termination, $\mathcal{W}2$ recommends changing a project according to the set $q' - q$. For example, in Figure **??**, 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 $\mathcal{W}2$ 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 $\mathcal{W}2$ 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 $\mathcal{W}2$ is not the only system that extends standard CBR with some generalization tools. Watson [**?**] 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 **??** and Equation **??** 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.

On termination, $\mathcal{W}$ recommends changing a project according to the set $q' - q$. For example, in Figure **??**, if $q' - q$ is $rely = 3$ then

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 [**?**], 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 11: Revising q to learn q'.

this treatment recommends that the best way to reduce the effort for this project is to reject $rely = 4$ *or* 5.

3.3 SEESAW

Since 2007, we have applied AI algorithms over parametric models of software development (based on COCOMO) [**?**] 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 **??**).

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:

$$prediction = model(P, T) \tag{3}$$

In the simplified COCOMO model of Equation **??**, 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).

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

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:

$$\arg\max_x \left(\overbrace{r_x \subseteq p}^{AI\ search}, \underbrace{t \subseteq T, value(model(r_x, t))}_{Monte\ Carlo} \right) \tag{5}$$

The intuition of Equation **??** was that, when faced with tuning variance like that seen in Figure **??**, 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 [**?**], 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:

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

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:

$$\forall x \in \{1..6\}\ SF_i = m_\beta(x - 6) \tag{7}$$

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 **??** and Equation **??**, but with different coefficients. Using experience from 161 projects [**?**], 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 **??**), 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 **??** using simulated annealing [**?**, **?**, **?**]. Subsequent work demonstrated that the recommendations found in this way did better than numerous standard process improvement methods [**?**]. Later implementations were based on a state-of-the-art theorem prover [**?**]. 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 **??** 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 **??**) 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).

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 **??**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

```
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 12: Pseudocode for SEESAW

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 [**?**], 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.

4. METHODOLOGY

4.1 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 [**?**] for examples of $\mathcal{W}$ running on a much broader set of inputs).

The inputs required for this study are:

- $\mathcal{W}$ 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 **??**.

Figure **??** 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 **??** to set its initial query q_0. SEESAW used Figure **??** to guide a set of simulations around its parametric models. For each case study, 1000 times, inputs were selected at random, constrained by Figure **??** (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 **??**). 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 **??**, 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.

	ranges			values	
project	feature	low	high	feature	setting
OSP: Orbital space plane	prec	1	2	data	3
	flex	2	5	pvol	2
	resl	1	3	rely	5
	team	2	3	pcap	3
	pmat	1	4	plex	3
	stor	3	5	site	3
	ruse	2	4		
	docu	2	4		
	acap	2	3		
	pcon	2	3		
	apex	2	3		
	ltex	2	4		
	tool	2	3		
	sced	1	3		
	cplx	5	6		
	KSLOC	75	125		
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		
	ranges			values	
project	feature	low	high	feature	setting
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		

Figure 13: The four NASA case studies. Numeric values *{1, 2, 3, 4, 5, 6}* map to *{very low, low, nominal, high, very high, extra high}*.

		Execution Time		
dataset	Instances	W	W2	W2 speedup
telecom1	18	0.07s	0.04s	1.6x
coc81	63	0.43s	0.08s	5.3x
nasa93	93	0.69s	0.10s	6.6x
china	500	7.37s	0.42s	10.8x

Figure 14: 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	W2	7%	48%	
kemerer	W	0%	44%	
miyazaki*	W2	75%	24%	
miyazaki	W	46%	45%	
telecom1	W	92%	23%	
telecom1	W2	81%	34%	
china	W2	34%	67%	
china	W	1%	36%	
finnish	W2	26%	28%	
finnish	W	18%	29%	

Figure 15: Performance of W2's Overlap relevancy filtering vs W's kth nearest-neighbor filtering for 5 unique datasets.

4.2 Comparing $\mathcal{W}$ to $\mathcal{W}2$

4.2.1 Effectiveness

Upon initial experimentation with $\mathcal{W}$, we often followed standard CBR methodology. For example, when deciding how to perform relevancy filtering, we chose the standard CBR practice of taking the euclidean distance from a defined project in n-dimensional space with n project features [**?**]. While this method performed well [**?**], the $O(n^2)$ runtime requirement prevented us from practically running $\mathcal{W}$ on very large datasets.

To resolve this, a simpler method for relevancy was devised. Instead of measuring relevancy based on the distance from a case to the project query's hypervolume, we decided to simply test for inclusion within this volume. The *overlap* of a case is simply the number of attributes that fall within the project query's ranges. Because our attributes must be discretized and often rely on qualitative metrics, large overlaps between a query and possible cases are common.

The performance of this new method is shown in Figure **??**. In all but one case, $\mathcal{W}2$ performs better. However, even when $\mathcal{W}2$ performs slightly worse, it still performs better than KNN in spread reduction. For the Miyazaki dataset, there exists a statistically significant difference (Mann-Whitney, 95% confidence level).

4.2.2 Execution Speed

Figure **??** shows the average execution times for $\mathcal{W}$ and $\mathcal{W}2$ across four datasets. While the original $\mathcal{W}$'s simplicity affords it runtimes of less than one second even on datasets with 93 instances (NASA93), the $O(n^2)$ kth nearest neighbor calculation becomes more significant when dealing with datasets as large as China with 500 instances. By replacing the $O(n^2)$ knn calculation with the linear *overlap* calculation, we maintain runtimes of less than half a second for larger datasets.

Runtimes of $\mathcal{W}2$

NASA93 Ground Stability

$\mathcal{W}$		$\mathcal{W}2$	
plex=3	85%	100%	plex=3
cplx=4	80%	85%	stor=3
stor=3	75%	55%	pmat=3
time=3	60%	45%	time=3
ltex=4	20%	20%	cplx=4
data=2	15%	10%	apex=3
pcap=3	15%	10%	rely=3
acap=3	10%	5%	pcap=3
apex=3	5%		
pmat=4	5%		
rely=4	5%		

Figure 16: How often a recommendation occured in 20 runs of $\mathcal{W}$ and $\mathcal{W}2$ for the Ground query on the NASA93 dataset. $\mathcal{W}2$ generally produces more stable recommendations than $\mathcal{W}$.

4.2.3 Stability

An unexpected cause for concern with $\mathcal{W}$ was some instability in its recommendations. After removing the nearest neighbor filtering, the *overlap* method for case relevancy filtering helps reduce this affect, as seen in Figure **??**. Note over 20 runs, $\mathcal{W}2$ *always* includes $plex = 3$ within q', whereas $\mathcal{W}$ omitted this recommendation for $\frac{3}{20}$ runs. $\mathcal{W}2$ also recommends only 8 different treatments compared to 11 for $\mathcal{W}$. While further study is needed regarding the stability of recommendations with $\mathcal{W}$, $\mathcal{W}2$ remains an improvement over $\mathcal{W}$.

5. RESULTS

5.1 $\mathcal{W}2$ vs SEESAW

Average median and spread results over the 30 trials are shown in Figure **??**. The fourth column in each group (labeled "Median Reduc") shows the relative change in effort, defect, months found by $\mathcal{W}2$ 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 single result.

The "Win" column indicates any member of a pair that was both statistically and significantly different. Note that for most pairs, the results are not statistically significantly different (Mann-Whitney, 95% confidence level).

Before commenting on SEESAW vs $\mathcal{W}2$, 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 48 experiments of Figure **??**, positive quality improvements were seen for $46/48 = 96\%$ experiments (the 2 exceptions are in the effort reductions of SEESAW for COC81 OSP2 and defect reductions of SEESAW for COC81 OSP).

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

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%	0%	
	months	SEESAW	17%	28%	
	months	W2	8%	17%	

Figure 17: Range of changes in median and spread generated by applying the recommendations of either $\mathcal{W}$ or SEESAW.

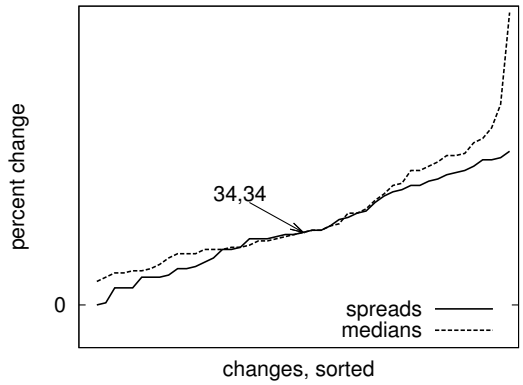

Figure 18: Range of changes in median and spread generated by applying the recommendations of either $\mathcal{W}$ or SEESAW. The median observed changes were (34, 34)% for (medians, spreads), respectively. For the sake of brevity, this graph ignores the -94% outlier value seen in OSP2 defects for SEESAW.

Another feature to note is that the median month optimizations for both SEESAW and $\mathcal{W}2$ are below 20% for $7/8 = 87.5\%$ of cases. That is, data sets 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.

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

Algorithm	Wins	Losses	Ties
W	6	0	18
SEESAW	0	6	18

Figure 19: 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.

Also, when the performance results were different, case-based reasoning did better than parametric modeling ($\frac{6}{24}$) sometimes spectacularly so, Observe the median reductions for NASA93 OSP (above the center divider of Figure **??**): $\mathcal{W}2$ offered median reductions of 72%, 69%, and 43% compared to SEESAW's 22%, 37%, 13% for defects, effort, and months respectively.

In summary, the simple case-based reasoning of $\mathcal{W}2$ performs just as well, or better, than SEESAW's elaborate parametric modeling.

5.2 $\mathcal{W}$ Performance Across Multiple Datasets

Because $\mathcal{W}$ makes no underlying model assumptions, we aren't limited to USC COCOMO for our performance 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://promisedata.org/data`. 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 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.

Effort reductions can be seen in Figure **??** and **??**. The chart in Figure **??** 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.

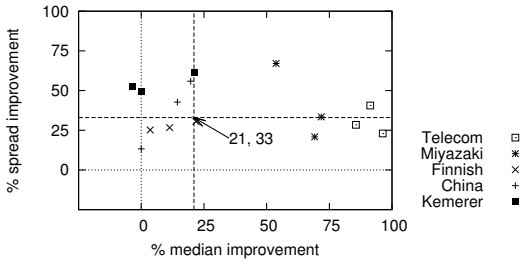

Figure 20: Plot showing the distribution of median and spread reductions in software effort for five unique datasets.

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

5.3 Comparing Drastic Changes to $\mathcal{W}$

	Drastic Change	Attribute Effects
1	Improve Process Maturity	pmat = 5
2	Improve Tools&Techniques	time = 3; stor = 3; pvol = 2; tool = 5; site = 6
3	Reduce Functionality	data = 2;

Figure 22: Examples of drastic changes to software projects.

NASA93 Ground					
Win	Goal	Change	Median Reduc	Spread Reduc	Reduction Quartiles 50%
	defects	ReduceFunct	64%	28%	
	defects	W	54%	32%	
	defects	Tools&Tech	51%	39%	
	defects	ProcMaturity	39%	73%	
	effort	ReduceFunct	62%	28%	
	effort	W	58%	32%	
	effort	Tools&Tech	46%	22%	
	effort	ProcMaturity	24%	76%	
	months	ReduceFunct	37%	16%	
	months	W	30%	16%	
	months	Tools&Tech	29%	26%	
	months	ProcMaturity	29%	33%	

Figure 23: 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.

Prior work considered explored impact of so-called *drastic* changes to software projects [**?**]. A drastic change occurs when the recommendation falls outside the defined projects ranges of a software project. In other words, when the recommended course of action is dramatic. For example, with the OSP NASA case study (Figure **??**), attempting to improve programmer language and tool experience (ltex) to 5 (very high, 6+ years experience), would be a drastic change as the maximum defined value for ltex is 4 (high).

To test this, $\mathcal{W}2$'s recommendations for the four NASA case studies (Figure **??**) in the NASA93 dataset were overridden with the drastic changes from Figure **??**. For 3 distinct drastic changes, $\mathcal{W}2$ attempted to apply the drastic changes until no improvement was measured, then reported the median effort, defects, and months for that change. Note that not all changes from [**?**] were applicable, due to a lack of extreme cases in the NASA93 dataset.

The results for these changes are reported in Figures **??**, **??**, **??**, **??**. Of the 12 comparisons, in only one case does $\mathcal{W}2$ perform better statistically and significantly better than the three drastic changes. However, in terms of median reductions, $\mathcal{W}$always performs in the top 50% of of cases. Most importantly, even when compared to extreme project recommendations, $\mathcal{W}$ is not constrained by the limited project attribute ranges allowed for its recommendations.

6. DISCUSSION

6.1 Search-based Software Engineering

Previously [**?**], we have explored the connection of SEESAW to search-based SE (SBSE) [**?**]. 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 [**?**]) but most often in the field of software testing [**?**]. Harman's writing inspired

NASA93 Flight					
Win	Goal	Change	Median Reduc	Spread Reduc	Reduction Quartiles 50%
	defects	ReduceFunct	64%	28%	
	defects	W	54%	32%	
	defects	Tools&Tech	51%	39%	
	defects	ProcMaturity	39%	73%	
	effort	ReduceFunct	62%	28%	
	effort	W	58%	32%	
	effort	Tools&Tech	46%	22%	
	effort	ProcMaturity	24%	76%	
	months	ReduceFunct	37%	16%	
	months	W	30%	16%	
	months	Tools&Tech	29%	26%	
	months	ProcMaturity	29%	33%	

Figure 24: 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.

NASA93 OSP					
Win	Goal	Change	Median Reduc	Spread Reduc	Reduction Quartiles 50%
	defects	W	61%	31%	
	defects	ProcMaturity	51%	26%	
	defects	ReduceFunct	46%	34%	
	defects	Tools&Tech	39%	32%	
*	effort	W	60%	28%	
	effort	ProcMaturity	51%	29%	
	effort	ReduceFunct	48%	36%	
	effort	Tools&Tech	47%	45%	
	months	ProcMaturity	31%	15%	
	months	W	30%	17%	
	months	Tools&Tech	25%	17%	
	months	ReduceFunct	25%	9%	

Figure 25: 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.

NASA93 OSP2					
Win	Goal	Change	Median Reduc	Spread Reduc	Reduction Quartiles 50%
	defects	W	64%	40%	
	defects	Tools&Tech	57%	24%	
	defects	ReduceFunct	50%	29%	
	defects	ProcMaturity	24%	38%	
	effort	ReduceFunct	63%	27%	
	effort	W	60%	45%	
	effort	Toolss&Tech	57%	36%	
	effort	ProcMaturity	14%	49%	
	months	W	35%	21%	
	months	Toolss&Tech	30%	15%	
	months	ReduceFunct	26%	12%	
	months	ProcMaturity	12%	21%	

Figure 26: 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.

us to try simulated annealing (SA) to search the what-ifs in untuned COCOMO models [**?**]. For quality optimization, however, we found that search methods taken from the constraint satisfaction literature out-perform SA [**?**].

6.2 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 [**?**]. 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 [**?**] 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.

7. CONCLUSION

We've demonstrated several improvements to our $\mathcal{W}$ algorithm with $\mathcal{W}2$. Namely:

- Optimization - $\mathcal{W}2$ runs faster, can be applied to larger datasets, and doesn't sacrifice performance.
- Explanation - $\mathcal{W}$ and $\mathcal{W}2$ are simple implementations of Contrast Set Learning, an easy to explain, intuitive learning process.
- Certification - $\mathcal{W}2$ performs as well or better than a multitude of software quality optimization techniques. Including parametric modelling techniques (SEESAW), drastic project changes, and multiple data sets based on various case descriptions.
- Application - $\mathcal{W}2$ can be applied to find locally learned recommendations that offer potentially unconsidered avenues for software quality optimization. $\mathcal{W}2$ can find alternatives to potentially dangerous situations such as the Brooks' Law example and drastic project changes.

In comparing the merits of a model-lite, case-based approach to a parametric one, advocates of reconstructive memory such as Barlett [**?**], Kolodner [**?**], or Shank [**?**] 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. [**?**]), 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 **??**. 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 **??**). 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 $\mathcal{W}2$ 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 **??**. $\mathcal{W}2$ extends standard CBR by learning an adaption of q, called q', that retrieves *better* quality examples. Based on the analysis of [**?**] and this paper, we recommend $\mathcal{W}2$ on several grounds:

- $\mathcal{W}2$ finds similar, or better, results than SEESAW (see Figure **??**).
- $\mathcal{W}2$ is simpler to code: 200 lines of AWK as opposed to the 5000 lines of LISP code used in SEESAW.
- $\mathcal{W}2$ is faster to run: the above experiments took seconds for $\mathcal{W}2$, but hours for SEESAW.
- $\mathcal{W}2$ is simpler to maintain since, in CBR, "maintenance" means nothing more than "add more cases".
- $\mathcal{W}2$ 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 [**?**].

Having said that, there is one situation where we'd recommend SEESAW over $\mathcal{W}$. Like all CBR systems, $\mathcal{W}2$ needs cases. If there is *no* local data, then SEESAW would be the preferred (only) option.

Firstly, there is insufficient evidence in this paper to make the conclusion that CBR *always* beats model-heavy methods like parametric models. Neverthless, 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 [**?**] 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.

Acknowledgments

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

APPENDIX

This appendix lists the minimum and maximum m values used for Equation **??** and Equation **??**. 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 [**?**], are:

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

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:

$$\begin{aligned} requirements &\left\{ \begin{array}{c} 0 \leq m_\gamma^+ \leq 0.112 \\ -0.183 \leq m_\gamma^- \leq -0.035 \end{array} \right. \\ design &\left\{ \begin{array}{c} 0 \leq m_\gamma^+ \leq 0.14 \\ -0.208 \leq m_\gamma^- \leq -0.048 \end{array} \right. \\ coding &\left\{ \begin{array}{c} 0 \leq m_\gamma^+ \leq 0.14 \\ -0.19 \leq m_\gamma^- \leq -0.053 \end{array} \right. \end{aligned} \tag{10}$$

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.