W2: A Simple, Flexible, Case-Based Recommendation Engine for Software Quality Optimization

Adam M. Brady

Thesis submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Computer Science

Tim Menzies, Ph.D., Chair
Cynthia Tanner
Tim McGraw, Ph.D.

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2011

Keywords: Data Mining, Toolkit, Software Defect Prediction, Bash, Awk, Scripting

Abstract

W2: A Simple, Flexible, Case-Based Recommendation Engine for Software Quality Optimization

Adam M. Brady

Researchers are drowning in choice as to how to 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). However, given many possible changes to a software project, which ones are recommended?

Two distinct strategies seek out this goal. Model-based methods seek a more general abstractions to describe software projects. Case-based methods instead seek local lessons based entirely from historical cases, referred here as *model-lite*. Given that case-based methods do not rely on an underlying model, they can be quickly adapted to a new domain and maintained by simply adding more cases.

$\mathcal{W}2$ is an case-based recommendation algorithm that seeks to improve software quality without constructing a general model. This thesis aims to justify the use of a simple, data-agnostic approach compared to a more sophisticated, data-specific model-based approach known as SEESAW.

We search for project recommendations within data from eight projects using various AI tools: six model-based methods and one case-based method, $\mathcal{W}2$. Results were assessed by comparing effort, defects, development time values in the raw data versus the subset of the data selected by those recommendations.

In the majority case, significantly large reductions on effort, defects and development time were achieved. Further, $\mathcal{W}2$ performed as well, or better, than any other methods in this study. While $\mathcal{W}2$ offers no conclusion on case-based vs model-based methods overall, our results show that simpler algorithms can be just as useful if not more so.

Dedication

For mom.

Acknowledgments

My utmost thanks to my family for their unwaivering praise and support, even if they still have no idea what I actually do. I also want to thank my advisor, Dr. Menzies, for showing me the exciting, bizarre misadventure that is academia.

Finally, I offer my gratitude to my fellow researchers and friends, for mitigating my sanity loss when teaching, classes, and research became too much.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

There are many ways a manager might *change*, and hopefully *improve*, their software development project. Some changes require *tools* such as using the new generation of functional programming languages or execution and testing tools [3] or automated formal analysis [24]. Other changes use *process improvement* techniques such as changing the organizational hiring practices, or a continual renegotiation of the requirements as part of an agile software development cycle [61].

Endres & Rombach [1] list dozens of *laws* of software engineering to justify a particular change to a project. If a manager proposed using *all* the laws, then senior management would most likely suggest they scale back their plans to just *a minimal set of most effective measures*.

Such analysis is critical early in a project's lifetime. Boehm's first law states that errors are most frequent in the design and requirements phases of development. Such errors are also more costly the later they are removed [1]. Software managers need tools that can suggest changes to a project given high-level characterizations early and effectively.

This thesis explores different ways for finding this minimal set of most effective changes to a project. Specifically, comparing *model-based* vs *case-based* methods. The difference between these two methods is as follows. Model-based methods develop a model via expert advice [19] or using automatic methods such as *data mining* [74]. Once built, the model can be used for "what-if" queries in order to assess possible changes to a project. For example:

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

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

Case-based methods, on the other hand, insert the "what-if" query into a n-dimensional space populated with historical project cases [32,34,35,62,69]. Unlike model-based methods, case-based methods do not require an underlying model. Rather, the immediate neighborhood of the "what-if" is somehow scored to find summary of those neighboring cases:

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

Previous work [17, 21, 41, 42, 44, 46, 47, 51, 57] tried combining model-based methods with AI tools to control thousands of "What-If" queries over COCOMO models. This thesis compares those model-based methods with "$\mathcal{W}2$", a novel case-based method. Given a "What-If" query that selects some set of similar projects, $\mathcal{W}2$ seeks a $treatment$ R_x, which finds the "better" parts of those similar projects within the dataset:

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

When compared to model-based methods:

- $\mathcal{W}2$ identified *similar* or *better* treatments.
- $\mathcal{W}2$ *was faster to run:* all the experiments in this thesis require 10 minutes with $\mathcal{W}2$, but days for using models.
- $\mathcal{W}2$ *was simpler to implement:* $\mathcal{W}2$'s 200 lines AWK replaces thousands of lines of the model-based LISP.
- $\mathcal{W}2$ *was simpler to maintain:* with case-based methods, "maintenance" implies "adding more cases".
- $\mathcal{W}2$ *was simpler to adapt to new domains:* $\mathcal{W}2$ do not require an underlying model and therefore it imposes no restrictions on the data being processed. It is more efficient and it can be quickly applied to more data sets.

Hence, this thesis recommends case-based methods like $\mathcal{W}2$ for identifying changes to software projects.

This result shows that the $\mathcal{W}2$ case-based method is superior to all the model-based methods explored in this study. This does not imply that learning changes to software projects is *always* best achieved using simple case-based methods. For example, the next release planning problem discussed in [56, 76] is a process problem of great complexity. For that task, the Pareto frontier optimization methods employed by (e.g.) Ruhe [56] is preferred to $\mathcal{W}2$.

Therefore, this thesis offers no *conclusion* on the relative merits of model-based versus case-based methods. Rather, it aims to show the relative benefits and performance potential of such case-based methods.

1.2 Statement of Thesis

$\mathcal{W}2$ represents a useful departure from standard model-based approaches for recommending changes to software projects. These experiments show that $\mathcal{W}2$ performs as well or better than more complex methods such as SEESAW, but offers greater flexibility in dataset applicability.

While this thesis cannot offer a conclusion on the whether to use model-based or case-based methods for improving software projects, $\mathcal{W}2$ provides a case-study in the usefulness of case-based methods.

1.3 Contribution of This Work

The work with $\mathcal{W}2$ and this thesis has 4 significant contributions:

1. $\mathcal{W}2$ enhances algorithms proposed by other researchers. Prior work on case-based methods found ways to *generate* estimates [30, 40, 68, 72]. $\mathcal{W}2$ shows that a small modification to standard case-based analysis can determine how to *change* an estimate.

2. $\mathcal{W}2$ outperforms the model-based methods described in [17, 21, 41, 42, 44, 46, 47, 51, 57] as well as earlier versions of $\mathcal{W}$ published in [12] and [11]. As discussed in §3, the $\mathcal{W}2$ algorithm presented here handles missing values better than early versions.

3. Recommendations from $\mathcal{W}2$ offer competing alternatives to standard "drastic" management recommendations.

4. Results show simple case-based methods can perform better than more complex model-based methods.

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

1.4 Document Structure

The following chapter lays out the related work when discussing model-based and case-based methods for improving software quality. Each approach is defined along with general benefits and drawbacks. Chapter 3 formally introduces the $\mathcal{W}2$ algorithm and its improvements, along with a worked example of recommending a change to a given project. Chapter 4 consists of the following experiments:

- A comparison between the original version of $\mathcal{W}$ and $\mathcal{W}2$. $\mathcal{W}2$ removes a $O(n^2)$ euclidean distance comparison for a linear time *overlap* calculation while retaining similar or better performance.

- $\mathcal{W}2$'s effectiveness in reducing estimated software effort across a range of arbitrary datasets

- A comparison of the decisions learned with $\mathcal{W}2$ to common drastic management decisions

- The stability of the conclusions recommended by $\mathcal{W}2$

Chapter 5 describes the experiment between model-based and case-based algorithms for software quality optimization. SEESAW is introduced as the best-performing straw-man example of model-based algorithms. $\mathcal{W}2$ is then compared to SEESAW in terms of estimated reductions in software effort, defects, and development months.

Chapter 6 continues the discussion of the comparison between model-based and case-based approaches. This chapter also discusses the application of $\mathcal{W}2$, including when it is useful and when other approaches may be more useful. Finally, Chapter 6 states concludes one the greater implications of tools like $\mathcal{W}2$ and the larger choice of how to recommend changes to software projects.

Chapter 2

Related Work

2.1 Software Estimation Research

Case-based software estimation such as *Case-Based Reasoning* (CBR) is a widely explored area in software engineering research [30,40,68,72]. Based on collective experience, when a manager sees an estimate, his/her immediate question is "how can I change that?". While the effort estimation literature describes many estimation methods (both model-based and case-based [26,30,36,37,40, 59,67,68,72]) in order to address manager's immediate concern, $\mathcal{W}2$ focuses on how to *change* estimates.

$\mathcal{W}2$ explores multiple goals such as reducing development effort *and* defects *and* the total calendar time to deliver the software. Instead, most other work in this area explores a single goal. For example, Pendharkar et al. [59] demonstrate the utility of Bayes networks in software effort estimation while Fenton and Neil explore Bayes nets and software defect prediction [18], neither of these teams links defect models to effort models. In addition, as mentioned above, these work focus much more on *prediction*, rather than on the subsequent problem of learning how to *change* those predictions.

2.2 Search-Based Software Engineering (SBSE)

Multi-goal optimization in Search-Based Software Engineering (SBSE) is well explored in the field [23]. SBSE employs optimization techniques from operations research and meta-heuristic search (for example in simulated annealing and genetic algorithms) in an attempt to hunt for near-optimal solutions. Harman [23] distinguishes AI search-based methods from those seen in standard numeric optimizations. Such optimizers offer settings to all controllables. This may result in needlessly complex recommendations since a repeated empirical observation is that many model inputs are contaminated or correlated in similar ways to model outputs [22]. Such contaminated or correlated variables can be pruned to generate simpler solutions that are easier and quicker to understand. For continuous variables, there are many work on feature selection [53] and techniques like principal component analysis [16] to reduce the number of dimensions reported by a data analysis. Some studies report that discrete AI methods perform better at reducing the size of the reported theory [22].

The SBSE approach can and has been applied successfully to many software engineering domains such as requirements engineering [25], but more commonly used in software testing [3]. Harman's work provides the inspiration to this study in an attempt to experiment simulated annealing for our model-based methods [44] (which performed worse than $\mathcal{W}2$).

2.3 Model: Benefits

High-level abstraction models represent and transmit common software patterns observed in multiple specific situations [20]. At a keynote address at PROSIM'05 Walt Scacchi noted that merely writing a model can clarify local business processes [60]. Executable software process models can be used for many purposes including but not limited to reducing the inspection effort at different stages of the software life cycle [49]. Even if a model lacks a sophisticated execution engine, it can still be used for what-if queries that are insightful to different business processes (e.g. see Boehm et al.'s what-if studies in Chapter Three of [10]).

Models can combine and summarize *both* expert insights *and* local data. Fenton [19] builds the general structure of his Bayes nets via workshops of business knowledge. The details of these structures are then tuned via local data. Elsewhere, Boehm reports the advantages of combining local data with model structures initialized via expert knowledge [14].

Another subtle advantage of models is data sharing. Schulz reports that organizations that are reluctant to share specific data, may be willing to share models (if those models do not reveal details from particular business sites) [65].

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

2.4 Model: Drawbacks

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

Another drawback with model-based tools is that they only accept data that conforms to the ontology of the model (i.e. use the input values of the model). If local data does not conform to that ontology, then the tool cannot be applied. For example, Figure 4.1 shows the data sets used in this study. Model-based methods can only process the two data sets that conform to the COCOMO ontology of Figure 2.1. On the other hand, the $\mathcal{W}2$ case-based method can process all of them.

Models need to be learned from data and collecting that data can be difficult due to the business sensitivity associated with the data as well as differences in how the metrics are determined, collected and archived. In many cases the required data is not archived at all. In our experience, for

	Definition	Low-end = {1,2}	Medium ={3,4}	High-end= {5,6}
Scale factors:				
flex	development flexibility	development process rigorously defined	some guidelines, which can be relaxed	only general goals defined
pmat	process maturity	CMM level 1	CMM level 3	CMM level 5
prec	precedentedness	we have never built this kind of software before	somewhat new	thoroughly familiar
resl	architecture or risk resolution	few interfaces defined or few risks eliminated	most interfaces defined or most risks eliminated	all interfaces defined or all risks eliminated
team	team cohesion	very difficult interactions	basically co-operative	seamless interactions
Effort multipliers				
acap	analyst capability	worst 35%	35% - 90%	best 10%
aexp	applications experience	2 months	1 year	6 years
cplx	product complexity	e.g. simple read-/write statements	e.g. use of simple interface widgets	e.g. performance-critical embedded systems
data	database size (DB bytes/SLOC)	10	100	1000
docu	documentation	many life-cycle phases not documented		extensive reporting for each life-cycle phase
ltex	language and tool-set experience	2 months	1 year	6 years
pcap	programmer capability	worst 15%	55%	best 10%
pcon	personnel continuity (% turnover per year)	48%	12%	3%
plex	platform experience	2 months	1 year	6 years
pvol	platform volatility $\left(\frac{frequency\ of\ major\ changes}{frequency\ of\ minor\ changes}\right)$	$\frac{12\ months}{1\ month}$	$\frac{6\ months}{2\ weeks}$	$\frac{2\ weeks}{2\ days}$
rely	required reliability	errors are slight inconvenience	errors are easily recoverable	errors can risk human life
ruse	required reuse	none	multiple program	multiple product lines
sced	dictated development schedule	deadlines moved to 75% of the original estimate	no change	deadlines moved back to 160% of original estimate
site	multi-site development	some contact: phone, mail	some email	interactive multimedia
stor	required % of available RAM	N/A	50%	95%
time	required % of available CPU	N/A	50%	95%
tool	use of software tools	edit,code,debug		integrated with life cycle

Figure 2.1: Features of the COCOMO model ontology.

example, after two years `promisedata.org` was only able to add 7 records to its NASA-wide software cost metrics repository [44]. Alternatively, open-source code repositories are a rich source of *product* information, but usually lack *process details* such as the descriptions of the applications experience of the developers.

Other researchers also have noted similar problems with collecting process data. Lowry [39] discusses the complexities involved in calibrating his software failure models. Those models require parameters that are clearly antiquated. For example, he mentions a commercial model-based cost estimation tool that requires a parameter that rates "the time it takes for a software development environment to respond to a keyboard input". When software was written on remote time-shared computers, this was an important factor. However, today it is irrelevant but it is kept in the model for backwards compatibility and because it was measured in the software projects on which the model was calibrated.

Baker [5] discusses another serious concerns with model calibration: *tuning instability*. Software construction is a very human-intensive process, therefore the data collected from that process is as varied as the humans building the code. Consider the following simplified COCOMO [10] model:

$$effort = a \cdot LOC^{b+pmat} \cdot acap \tag{2.1}$$

The equation presents COCOMO's core assumption that software development effort is exponential on software size. In this equation, a and b control the linear and exponential inferences (respectively) on model estimates; while $pmat$ (process maturity) and $acap$ (analyst capability) are project choices articulated by managers. Equation 2.1 contains only two features ($acap, pmat$) and a full COCOMO model contains a set of project descriptors as shown in Figure 2.1.

Baker [5] learned values of (a,b) for a full COCOMO model using Boehm's local calibration method [7] from 300 random samples of 90% of the available project data. The ranges varied widely:

$$(3.2 \leq a \leq 9.4) \wedge (0.8 \leq b \leq 1.12) \tag{2.2}$$

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

The plot in Figure 2.2 shows the b values learned from twenty 66% samples (selected at random) of the NASA93 data set from the PROMISE repository. Prior to learning, training data was *linearized* in the manner recommended by Boehm (x was changed to $log(x)$; for details, see [48]). 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.

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 $-2 \leq b \leq 8$.

- Consequently, nine of the coefficients in Figure 2.2 jump from negative to positive.

We have seen instability in other datasets, including the COC81 data used by Boehm to derive the general form of Equation 2.1 [48]. This is an troubling observation.

In summary, model-based methods can suffer from:

- Inappropriate extrapolations;

- Ontology restrictions;

- Untamed variance inside the models

Hence the need explore alternative methods.

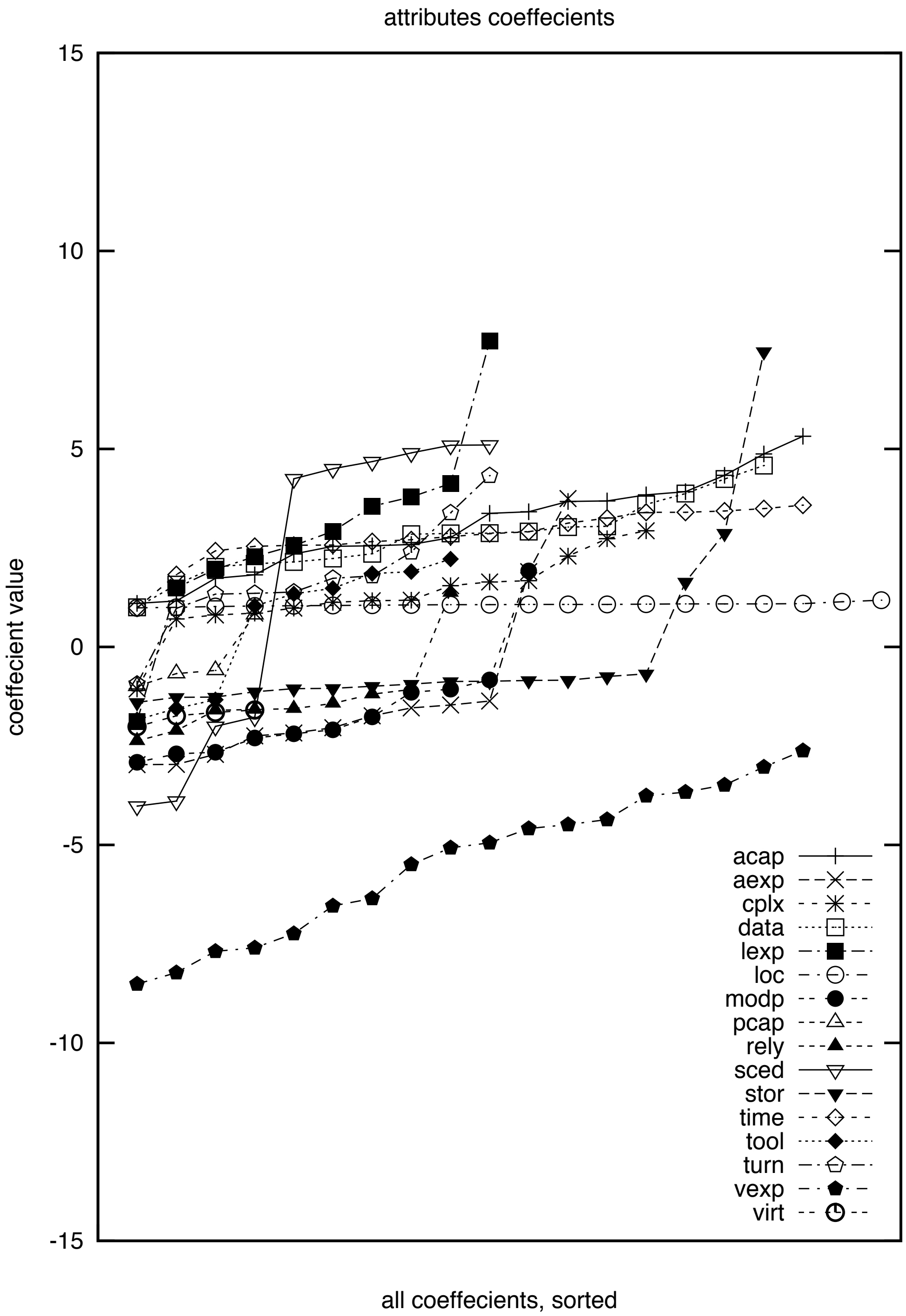


Figure 2.2: COCOMO 1 effort multipliers, and the sorted coefficients found by linear regression from twenty 66% sub-samples (selected at random) from the NASA93 PROMISE data set; from [48].

Chapter 3

The $\mathcal{W}2$ Algorithm

$\mathcal{W}2$ is a simple, data-agnostic case-based reasoning tool for software quality optimization. Given a dataset of historical cases and a range of controllable attributes within that space, $\mathcal{W}2$ recommends changes that offer the best utility within that space.

In the case of software projects, $\mathcal{W}2$ recommends changes that best improve estimated software quality given a description of a potential new project.

3.1 Case-Based Reasoning

An opposite approach to model-based methods is case-based reasoning (CBR), also referred to as instance-based reasoning. In CBR, there are no universally-applicable 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 [6].*

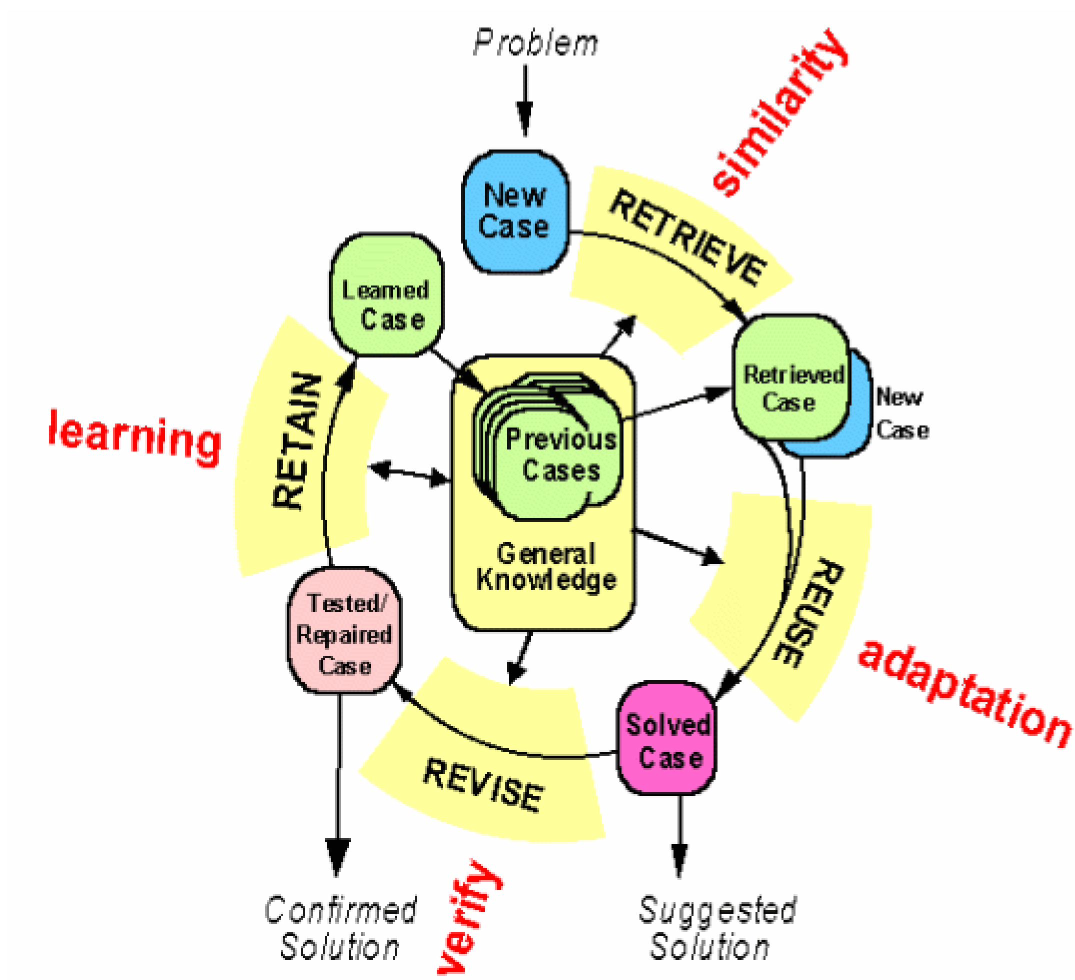


Figure 3.1: Four steps of case-based reasoning, from `http://www.peerscience.com/intro_cbr.htm`.

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 [38].

In AI research, Janet Kolodner [33] 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.

Inference using case-based reasoning is usually characterized [2] 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.

Having verified the results from a 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 [63].

(Note that we call CBR *model-lite*, but not *model-free*. For more on this distinction, see §6.2.)

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 $\mathcal{W}2$’s case-based reasoning.

3.2 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 [41], the ensuring rare attributes are weighted appropriately:

$$like = \frac{freq(x|best)^2}{freq(x|best) + freq(x|rest)} \tag{3.1}$$

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 3.1 is to log the odds ratio between an attribute appearing in *best* rather than *rest* [55].

3.3 The $\mathcal{W}2$ Algorithm

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 [31]. We offer a statement on performance between $\mathcal{W}$ and $\mathcal{W}2$'s in the §4.2 and Figure 4.4.

$\mathcal{W}2$ 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

```
@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 3.2: The Brooks' Law Query for the NASA93 dataset in COCOMO II format.

access to tools, then those constraints can be expressed in the syntax of Figure 3.11.

$\mathcal{W}2$ is easily demonstrated visually. Figure 3.2 demonstrates a query representing a project query q involving Brooks' Law [13] 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,

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

(other cases omitted)

Figure 3.3: RELEVANT= cases nearest to $context_1$.

etc.

The query in Figure 3.2 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:

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

Figure 3.4: *Best* (top) & *rest* (bottom).

3.3.1 Relevancy Filtering

From Training, 20 cases are selected with the highest total *overlap* with the project query (Figure 3.3). 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.

3.3.2 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 3.5). For datasets with multiple goals, such as the NASA93 and COC81 datasets that contain project effort, defects, and months, a utility function

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

Figure 3.5: Rank with Equation 3.1.

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.

3.3.3 Contrast Set Generation

Changes to q are ranked according to equation 3.1. This sorted order S defines a set of candidate q' queries that use the first *i-th* entries in S (Figure 3.4):

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

In the Brooks' Law example, $\mathcal{W}2$ learns that *pmat=3* scores the highest for reducing develop-

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

Figure 3.6: A $K_1 = 20$ neighborhood of $context_1$ (NASA93ii train set).

ment effort. This is the last half CBR *reuse* (or adapt) step.

At this point, $\mathcal{W}2$ has created a ranked list of recommendations that best drive q towards more desirable utility measures (Figure 3.5). 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.

3.3.4 Estimating Impact

According to Figure 3.1, 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 3.10.

On termination, $\mathcal{W}2$ recommends changing a project according to the set $q' - q$. For example,

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

Figure 3.7: All rows of Figure 3.6 satisfying $R_1 : pmat = 3$.

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

Figure 3.8: The testing set with all cases not containing $pmat = 3$ removed.

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

Figure 3.9: 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 $pmat = 3$.

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 [68], 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 3.10: Revising q to learn q'.

in Figure 3.11, 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 case-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 [73] 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 3.10 and Equation 3.1 is only a few dozen lines of code. That is, $\mathcal{W}2$ is such a small extension to standard CBR that it would be somewhat pedantic to declare that it is not "real" CBR.

On termination, $\mathcal{W}2$ recommends changing a project according to the set $q' - q$. For example,

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

Figure 3.11: $\mathcal{W}2$'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)

in Figure 3.11, 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.

3.4 Measuring Performance

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

- All our measures are taken from the test set.
- The *asis* values are from the neighborhood of the *context*; e.g. the $effort$ column.
- The *tobe* values are from the cases selected by a treatment; e.g. the $effort$ column.
- The *median* of a distribution is the 50-th percentile of the sorted values in that distribution.
- The *spread* of a distribution is the (75-25)th percentile of the sorted values.
- The *improvement* from $a = asis$ to $t = tobe$ is $100 * (a - t)/a$. *Larger* improvements are better.

For example, consider $pmat = 3$:

- Without the $pmat = 3$ restriction, the median and spread in the test set are 235 and 633 months, respectively.

- With $pmat = 3$, the median and spread of projects similar to $context_1$ are 81 and 353 months (see Figure 3.7).
- The observed improvement in the median is hence 66%.
- The observed improvement in the spread is hence 44%.

Chapter 4

Experiments with $\mathcal{W}2$

4.1 Datasets and Project Descriptions

Dataset	Cols	Rows	Notes	Measures
Kemerer	7	15	Large business applications	effort
Telecom	3	18	U.K. telecom enhancements	effort
Finnish	8	38	Finnish IS projects	effort
Miyazaki	8	48	Japanese COBOL projects	effort
COC81ii	26	63	NASA projects	effort, time, defects
NASA93ii	26	93	NASA projects	effort, time, defects
China	17	499	Chinese software projects	effort
		Total: 774		

Figure 4.1: Seven data sets from `promisedata.org/?cat=14`: *effort* is total staff person-months; *time* is calendar time (start to stop); *defects* represents the number of delivered defects.

Because $\mathcal{W}2$ makes no underlying model assumptions, we aren't limited to the COCOMO II ontology for our performance evaluations. However, in order to compare $\mathcal{W}2$'s case-based methodology with SEESAW's model-based methodology, these comparisons must be made within the same ontology. For the following experiments, any comparison with SEESAW will use the NASA93 and COC81 datasets.

Both NASA93 and COC81 initially contained only effort data in the original COCOMO format. However, using Boehm's COQUALMO defect prediction we have converted this data into COCOMO II format, which contains estimated total defect numbers as well as estimated completion time in months. COCOMO II also breaks down the project attributes into both scale factors and effort multipliers, outlined earlier in Figure 2.1.

The other datasets use for these experiments is detailed in Figure 4.1. For non-COCOMO data, the reduction goal is the same: software effort in person-months. These datasets include the following:

- Enhancements to a U.K. telecommunications product;
- Projects collected by Miyazaki et al [54];
- Finnish Information Systems projects;
- A large dataset of Chinese software projects;
- Large COBOL projects, collected by Kemerer [27].

The format of this data is highly varied and includes number of basic logical transactions, query count and number of distinct business units serviced.

Project descriptions are also needed to define the space of controllable options. For the non-COCOMO data sets, we did not have access to specific case studies like Figure 5.1. Hence, these results are based on *contexts* developed as follows:

- One project contained all possible project values as $controllables$
- The remaining two projects featured randomly chosen ranges with ranges half of the possible values.

Data and project descriptions for the NASA case studies defined in Figure 5.1 are shown in §B. A project description file for a non-COCOMO dataset is shown in Figure 4.2.

```
@project china-Proj2
@attribute ?AFP vl lo md
@attribute ?Input md hi vh
@attribute ?Output vl lo md
@attribute ?Enquiry md hi vh
@attribute ?File vl lo md
@attribute ?Interface hi vh
@attribute ?Added vl lo md
@attribute ?Changed md hi vh
@attribute ?Deleted vl lo md
@attribute ?PDR_AFP md hi vh
@attribute ?PDR_UFP md hi vh
@attribute ?NPDR_AFP vl lo md
@attribute ?NPDU_UFP vl lo md
@attribute ?Resource md hi
@attribute ?Dev.Type vl
@attribute ?Duration vl lo md hi
```

Figure 4.2: Example project *controllable* file for Chinese software projects after discretization. Ranges were assigned randomly for this project. A "?" represents a controllable feature. If an attribute range isn't specified in the project, it is ignored.

4.2 Experiment: $\mathcal{W}2$ vs $\mathcal{W}$

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 [68]. While this method performed well [11], 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 4.4.

		Execution Time		
dataset	Cases	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	499	7.37s	0.42s	10.8x

Figure 4.3: 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 (499 cases).

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%	

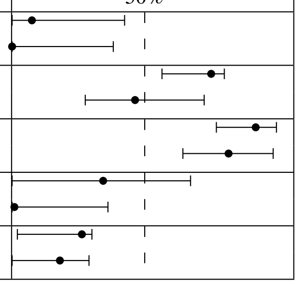


Figure 4.4: Performance of W2's Overlap relevancy filtering vs W's kth nearest-neighbor filtering for 5 unique datasets.

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

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).

4.3 Experiment: $\mathcal{W}2$'s Performance Across Multiple Datasets

To demonstrate the effectiveness of $\mathcal{W}2$ in any data environment, we offer median reductions for effort reduction for five arbitrary datasets from the PROMISE data repository. The model-agnostic simplicity of $\mathcal{W}2$ 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 4.5 and 4.6. The chart in Figure 4.5 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.

The other two queries were generated by randomly choosing 50% of each attribute values from either the lower, middle, or upper ranges for each project descriptor. For these experiments, the *values* function was just "reduce effort" (the next experiment will explore other results on COCOMO-related data, that tries to reduce effort *and* defects *and* calendar months).

There are three noteworthy aspects of the Figure 4.6 results:

- All the points in that figure are positive; i.e. improvements were seen in all cases.

- The dotted lines show the 50% percentile range of the results: half that results had at least

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

56% and 73% improvement in median and spread.

- There is no evidence that $\mathcal{W}2$ has problems with smaller data sets. The two smallest examples processed by $\mathcal{W}2$ are Kemerer and Telecom containing 15 and 18 examples each. The minimum improvements seen, even for these small data sets, are 55% (in both median and spread).

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

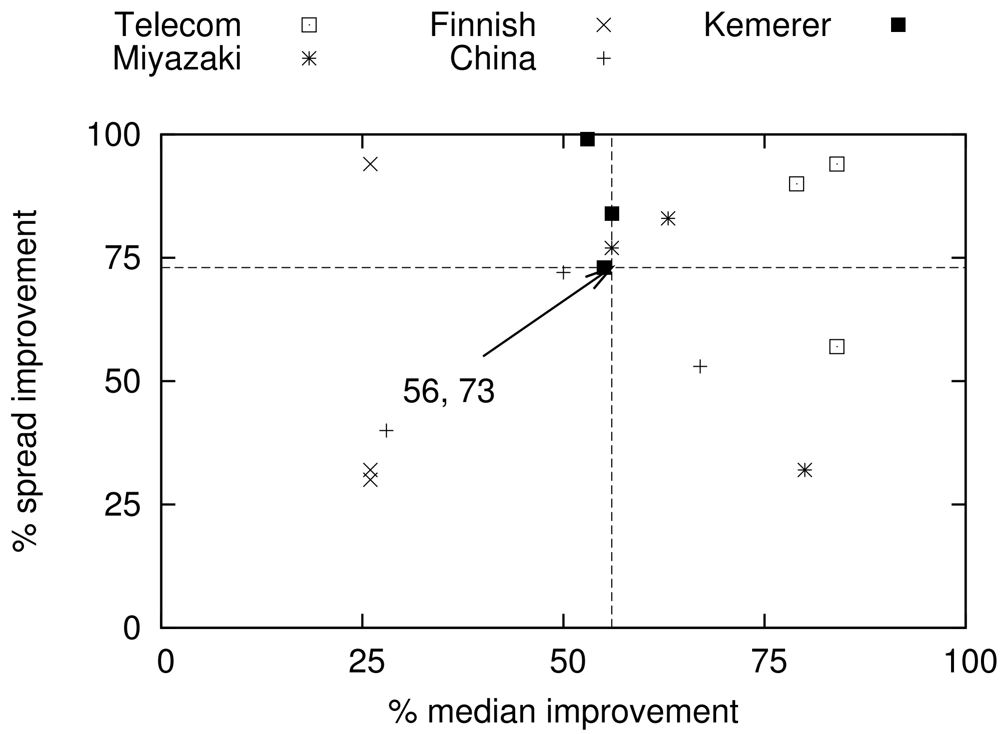


Figure 4.6: Effort results for five non-COCOMO datasets.

Optimizing for $N = 3$ goals is a harder task than just the $N = 1$ goal of Figure 4.6. Hence, those the improvements seen in those examples will be more modest (around 20%).

4.4 Experiment: Intra- and Inter-Project Stability

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

To test this premise, we generated a report of what treatments were found under different conditions. Figure 4.8 shows the results of $\mathcal{W}2$'s $Step7$ (prune all treatments that do appear in less than 50% of 20 repeated trials). The left-hand column of Figure 4.8 shows the four *values* function used in that study:

1. *Defects* aims at reducing just defects;
2. *Effort* aims at reducing just effort;
3. *Months* aims at reducing a project's total calendar time.
4. *All* refers to Equation 5.7; i.e. try to decrease effort *and* development time *and* number of defects;

The last of these is a multi-objective function while the rest strive to optimize one objective without concern to the others. Figure 4.8 shows that, in any row, the conclusions reached by $\mathcal{W}2$ are stable (i.e. appear at high frequency, across 20 random selections of $train : test$). That is, $\mathcal{W}2$'s results exhibit $intra-$project conclusion stability (when the $context$ and $values$ function are held constant). For project managers, this is good news since it shows that their data contains clear signals on how

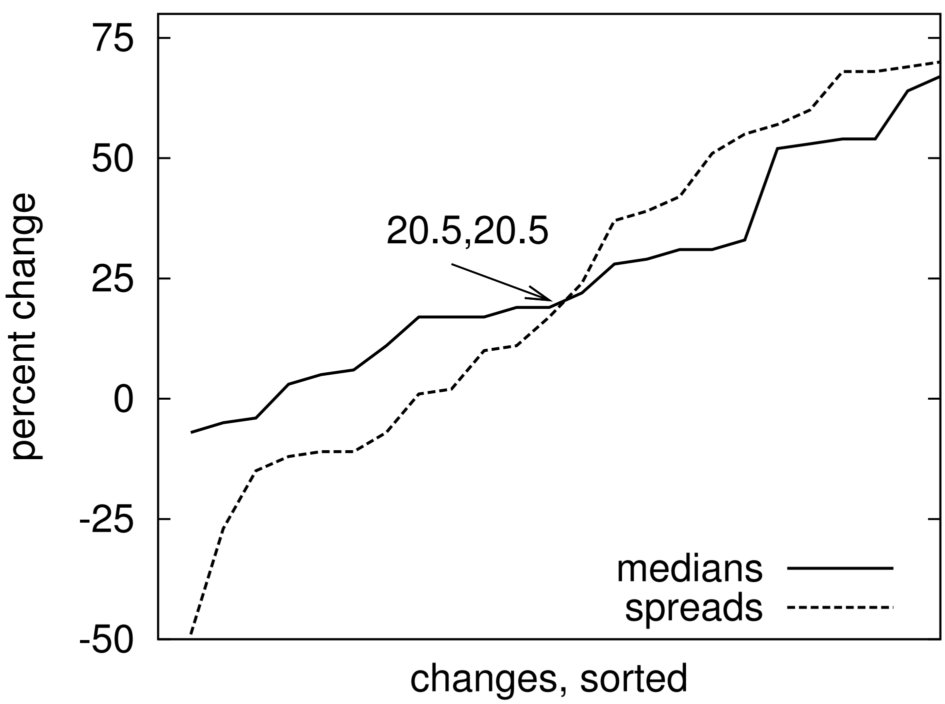


Figure 4.7: Range of changes in median and spread generated by applying the recommendations of $\mathcal{W}2$. The median observed changes were (20.5, 20.5)% for (medians, spreads), respectively.

Stability Comparison for NASA93ii FLIGHT											
goal	*acap=3*	*apex=3*	*apex=5*	*ltex=4*	*pmat=3*	*pmat=4*	*rely=3*	*sced=2*	*stor=3*	*time=3*	*changes*
defects					90				65	75	3
effort					70					90	2
months					70				60	85	3
all					75				70	85	3
Stability Comparison for NASA93ii GROUND											
goal	*acap=3*	*apex=3*	*apex=5*	*ltex=4*	*pmat=3*	*pmat=4*	*rely=3*	*sced=2*	*stor=3*	*time=3*	*changes*
defects					80				50		2
effort					60					75	2
months					75		50			75	3
all					85					80	2
Stability Comparison for NASA93ii OSP											
goal	*acap=3*	*apex=3*	*apex=5*	*ltex=4*	*pmat=3*	*pmat=4*	*rely=3*	*sced=2*	*stor=3*	*time=3*	*changes*
defects	95		70					50			3
effort	80	70						70			3
months						55		80			2
all	50							85	60		3
Stability Comparison for NASA93ii OSP2											
goal	*acap=3*	*apex=3*	*apex=5*	*ltex=4*	*pmat=3*	*pmat=4*	*rely=3*	*sced=2*	*stor=3*	*time=3*	*changes*
defects				60		75		95			3
effort				75		55		80			3
months				70		80		90			3
all				80		90		100			3

Figure 4.8: Recommendation frequency across 20 runs of $\mathcal{W}2$ for reducing individual goals ($defects$, $effort$, or *months*) as well as all goals at once (*all*).

to best change their particular project in order to achieve particular goals.

However, Figure 4.8 also shows that if the *context* is changed (from generalized FLIGHT systems to a specific flight system like OSP), then the recommended changes are very different. Similarly, the OSP results show that altering the *values* function also dramatically changes recommendations.

Menzies & Shull [50] report that many SE papers conclude that what is “best” for one project may not be “best” for another. For example, Zimmermann studied 629 pairs of software development projects [70]. In only 4% of those hundreds of pairs was a defect prediction model learned from one project useful on another. When such *inter*-project conclusion instability exists, then tools like $\mathcal{W}2$ are essential since it is best to learn changes that are tuned to the specifics of particular projects (like OSP & OSP2) rather than on generalized descriptions of software (like FLIGHT & GROUND).

4.5 Experiment: Comparing Drastic Changes to $\mathcal{W}2$

Prior work considered explored impact of so-called *drastic* changes to software projects [46]. 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 5.1), 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 5.1) in the NASA93 dataset were overridden with the drastic changes from Figure 4.9. 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 [46] were applicable, due to a lack of extreme cases in the NASA93 dataset.

The results for these changes are reported in Figures 4.11, 4.10, 4.12, 4.13. 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, $\mathcal{W}2$ always performs in the top 50% of of cases. Most importantly, even when compared to extreme project recommendations, $\mathcal{W}2$ is not constrained by the limited project attribute ranges allowed for its recommendations.

	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 4.9: Examples of drastic changes to software projects.

Rank	Goal	Change	Median Reduc	Spread Reduc	Reduction Quartiles 50%
1	defects	ReduceFunct	64%	28%	
1	defects	W	54%	32%	
1	defects	ToolsTech	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	ToolsTech	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	ToolsTech	29%	26%	
1	months	ProcMaturity	29%	33%	
1	months	ReduceQuality	0%	98%	
2	months	RelaxScedule	-3%	16%	

Figure 4.10: Comparing defect, effort, and month estimation reduction percentages ($100 * \frac{initial-final}{intial}$ of drastic business decisions vs $\mathcal{W}$'s recommendations for the 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	ToolsTech	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	ToolsTech	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	ToolsTech	29%	26%	
1	months	ProcMaturity	29%	33%	
1	months	ReduceQuality	0%	98%	
2	months	RelaxScedule	-3%	16%	

Figure 4.11: Comparing defect, effort, and month estimation reduction percentages ($100 * \frac{initial-final}{intial}$ of drastic business decisions vs $\mathcal{W}$'s recommendations for the Flight case study.

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 4.12: Comparing defect, effort, and month estimation reduction percentages ($100 * \frac{initial-final}{intial}$ of drastic business decisions vs $\mathcal{W}$'s recommendations for the 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 4.13: 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.

Chapter 5

Model-Based vs. Case-Based Algorithms

5.1 Model-based Case Studies

Since the presented model-based methods are built around the COCOMO-suite, we must use COCOMO data and *contexts* written in the COCOMO ontology. An example of such data is the NASA93 dataset found in §B and is available at `promisedata.org`. Figure 5.1 shows some real-world *context* and *control* information taken from a debrief of some NASA program managers:

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

The un*control*able column in Figure 5.1 shows project features that cannot be changed. For example in project OSP, the required reliability is fixed at $rely = 5$. On the other hand, the *low* and

	context				
	*control*able			un*control*able	
project	feature	low	high	feature	setting
OSP: Orbital space plane	prec	1	2	data	3
	flex	2	5	pvol	2
	resl	1	3	rely	5
	team	2	3	pcap	3
	pmat	1	4	plex	3
	stor	3	5	site	3
	ruse	2	4		
	docu	2	4		
	acap	2	3		
	pcon	2	3		
	apex	2	3		
	ltex	2	4		
	tool	2	3		
	sced	1	3		
	cplx	5	6		
	KSLOC	75	125		
JPL flight software	rely	3	5	tool	2
	data	2	3	sced	3
	cplx	3	6		
	time	3	4		
	stor	3	4		
	acap	3	5		
	apex	2	5		
	pcap	3	5		
	plex	1	4		
	ltex	1	4		
	pmat	2	3		
	KSLOC	7	418		
OSP2	prec	3	5	flex	3
	pmat	4	5	resl	4
	docu	3	4	team	3
	ltex	2	5	time	3
	sced	2	4	stor	3
	KSLOC	75	125	data	4
				pvol	3
				ruse	4
				rely	5
				acap	4
				pcap	3
				pcon	3
				apex	4
				plex	4
				tool	5
				cplx	4
				site	6
JPL ground software	rely	1	4	tool	2
	data	2	3	sced	3
	cplx	1	4		
	time	3	4		
	stor	3	4		
	acap	3	5		
	apex	2	5		
	pcap	3	5		
	plex	1	4		
	ltex	1	4		
	pmat	2	3		
	KSLOC	11	392		

Figure 5.1: Contexts of 4 case studies. {*1, 2, 3, 4, 5, 6*} map to {*very low, low, nominal, high, very high, extra high*}.

high ranges in that figure define the space of possible changes to that project. For instance, the reliability of flight software varies from 3 (nominal) to 5 (very high).

5.2 SEESAW

Since 2007, researchers at WVU have applied AI algorithms over parametric models of software development (based on COCOMO) [44] to implement quality optimizers. This is a a challenging task since they 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 2.2).

In order to address this challenge, one needs 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:

$$prediction = model(P, T) \tag{5.1}$$

In the simplified COCOMO model of Equation 5.2, 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).

$$effort = a \cdot LOC^{b+pmat} \cdot acap \tag{5.2}$$

Based on the definitions of the COCOMO model, the ranges of the project attributes are:

$$P = 1 \leq (pmat, acap) \leq 5 \tag{5.3}$$

Further, the cone of uncertainty associated with a particular project p 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 they 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:

$$\arg\max_x \left(\overbrace{r_x \subseteq p}^{AI\ search}, \underbrace{t \subseteq T, value(model(r_x, t))}_{Monte\ Carlo} \right) \tag{5.4}$$

The intuition of Equation 5.4 was that, when faced with tuning variance like that seen in Figure 2.2, one 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. Shown in Figure 2.1, 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 [10], 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:

$$\forall x \in \{1..6\}\ EM_i = m_\alpha(x-3)+1 \tag{5.5}$$

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

One 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:

$$\forall x \in \{1..6\}\ SF_i = m_\beta(x-6) \tag{5.6}$$

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 5.5 and Equation 5.6, but with different coefficients. Using experience from 161 projects [10], one 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 5.4), all we need to do is select m values at random from the min/max m values ever seen.

Initially, prior work implemented the AI search of Equation 5.4 using simulated annealing [43, 44, 47]. Subsequent work demonstrated that the recommendations found in this way did better than numerous standard process improvement methods [46]. Later implementations were based on a state-of-the-art theorem prover [21]. 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 5.2 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 5.2) 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 5.2 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

```
function run (AllRanges, ProjectConstraints) {
  OutScore = -1
  P = 0.95
  Out = combine(AllRanges, ProjectConstraints)
  Options = all Out features with ranges low < high
  while Options {
    X = any member of Options, picked at random
    {Low, High} = low, high ranges of X
    LowScore = score(X, Low)
    HighScore = score(X, High)
    if LowScore < HighScore
      then Maybe = Low; MaybeScore = LowScore
      else Maybe = High; MaybeScore = HighScore
    fi
    if MaybeScore < OutScore or P < rand()
      then delete all ranges of X except Maybe from Out
      delete X from Options
      OutScore = MaybeScore
    fi
  }
  return backSelect(Out)
}
function score(X, Value) {
  Temp = copy(Out) ;; don't mess up the Out global
  from Temp, remove all ranges of X except Value
  run monte carlo on Temp for 100 simulations
  return median score from monte carlo simulations
}
```

Figure 5.2: Pseudocode for SEESAW

difficult to maintain.

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

- In the three years since our first paper [44], 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.

5.3 Five Additional AI Model-Based Algorithms

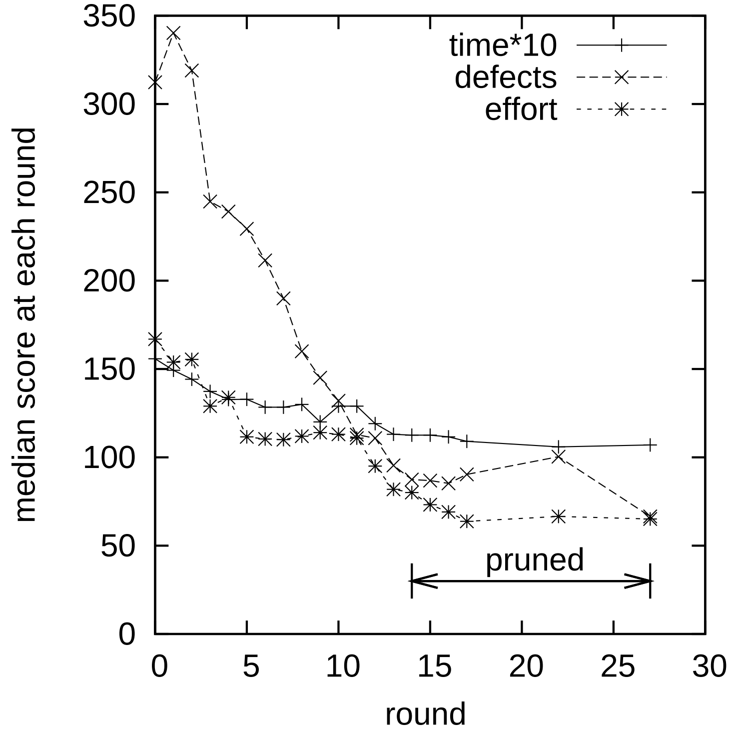
The case studies of Figure 5.1 can be used to assess how well different AI algorithms can find changes to software projects. For example, a typical Simulated Annealing (SA) run explores 10,000 variants on some solution [29]. A side-effect of that run is 10,000 sets of inputs, each scored with the *value* function of Equation 5.7.

$$value = 1 - \left(\sqrt{Effort^2 + Defects^2 + Time^2}/\sqrt{3}\right) \tag{5.7}$$

Our tool classified the outputs into the 90% *rest* and the 10% *best* seen during the run of the SA. All the ranges from all the features were then ranked according to how much more frequently they appeared in *best* than *rest*. A *forward select* was then called using the first $1 \ldots x$ ranked items. Figure 5.3 shows the treatment R_x at any x value is the conjunction of ranges observed between 1 to x (see the table at the bottom of that figure). The y axis scores show median results in 100 runs of COCOMO/COQUALMO, after imposing the treatment. The "pruned" range of that figure shows the results of a *back select* that worked backwards over the forward select ordering, deleting any item x whose distribution of values was statistically insignificantly different to $x-1$. SA's final recommendation was the treatment $1 \leq x \leq 13$. The *improvement* generated by that treatment can be seen by comparing the values at $x = 0$ to $x = 13$.

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

SA is just one way to generate a treatment. For our AI model-based methods, we explored five others. Given a random selected treatment, *MaxWalkSat* tries n modifications to randomly selected features [66]. Sometimes (controlled by the α parameter), the algorithm chooses the range that minimizes the value of the current solution. Other times (at probability $1-\alpha$), a random range



Decisions made from round=1 to round=13:

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

Figure 5.3: Example of SA's forward and back select.

is chosen for the feature. After N retries, the best solution is returned. Our implementation used $n = 50$, $\alpha = 0.5$, and $N = 10$.

ISAMP is a fast stochastic iterative sampling method that extends a treatment using randomly selected ranges. The algorithm follows one solution, then resets to try other paths (our implementation resets 20 times). The algorithm has proved remarkably effective at scheduling problems, perhaps because it can rapidly explore more of the search space [15]. To avoid exploring low-value regions, our version of ISAMP stores the worst solution observed so far. Any conjunction whose *value* exceeds that of the worst solution is abandoned, and the new "worst value" is retained. If a conjunction runs out of new ranges to add, then the "worst value" is slightly decreased. This ensures that consecutive failing searches do not permanently raise the "worst value" by an overly permissive value.

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

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

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

- f was estimated as the percentage of the project descriptors with ranges in the current branch;
- g was estimated using $1 - Equation$ 5.7 (i.e. distance to the utopia of no effort, no development time, and no defects).

5.4 Comparisons of AI Model-based Methods

For each case study of Figure 4.1, each algorithm was run 20 times (guided by the *value* function of Equation 5.7). Separate statistics were collected for the defects/effort/time predictions seen at the policy point in the 20*4 trials. The *top-ranked* algorithm(s) of Figure 5.4 had statistically different and lower defects/effort/time predictions than any other algorithm(s).

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

Figure 5.4: Number of times algorithms were top-ranked (largest is 4: i.e. one for each Figure 5.1 case study).

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

From Figure 5.4, the worst algorithms are MaxWalkSat and ISAMP and the best algorithms are SEESAW and BEAM. The performance of these best algorithms is sometimes equivalent (e.g., in *time*, both algorithms achieved an equal number of top ranks). However, BEAM is not recommended:

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

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

5.5 Model vs. Case-Based Methods

SEESAW requires models in the COCOMO format so for our comparisons, we restrict ourselves to data in that format. $\mathcal{W}2$ used the historical cases from the NASA93ii and COC81ii datasets. These data sets all have the features defined by Boehm [7]; e.g. analyst capability, required software reliability, and use of software tools. Originally collected in the COCOMO-I format, JPL business experts have translated them from their original COCOMO format to COCOMOII.

Both SEESAW and $\mathcal{W}2$ guided their search using Equation 5.7 and the four *context*s of §5.1. SEESAW used those *context*s to guide their "what-if" queries around its COCOMO/COQUALMO models. $\mathcal{W}2$ took those *contexts* then applied the seven step procedure described above to NASA93ii and COC81ii. Recall that, in those steps, some R_x was assessed on projects similar to the *context* in a *test* set; i.e. all the cases in the *context*'s neighborhood. Our comparison rig studied that same *test* neighborhood using SEESAW and $\mathcal{W}2$. We say that $rows_1$ and $rows_2$ are the rows selected from the neighborhood after applying SEESAW's or $\mathcal{W}2$'s recommendations (and by "apply", we mean reject any row that contradicts the ranges in the recommendation). From $rows_i$, we applied Equation 5.7 to find $values_i$.

The are shown in Figure 5.6, divided into the defect, effort, months changes see in GROUND, FLIGHT, OSP2 and OSP. In all, we show 24 comparisons:

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

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

Figure 4.7 shows the sorts the median and spread improvements seen from the Figure 5.6 results. Note that rarely were the changes to the median less than zero. In the majority of cases, W2's median and spread improvements were positive (an expected value of 20.5; sometimes ranging over 50%). While occasionally the spread degraded sharply (down to 50% *worse*), such cases were uncommon: note that in only 10% of our results were the spread changes below -15%. Also, all the cases where $\mathcal{W}2$ had poor spread results were in the COC81ii data set which, as discussed below, is a data set with certain special features.

Win	Goal	Treatment	50% percentile (median) $a =$ as is	$t =$ to be	(75-25)th percentile (spread) $a =$ as is	$t =$ to be	Median improv. $\frac{a-t}{a}$	Spread improv. $\frac{a-t}{a}$
			NASA93ii Flight					
	defects	SEESAW	1276	626	3737	2311	51%	38%
	defects	W	2042	1688	3992	2501	17%	37%
	effort	SEESAW	159	72	378	192	55%	49%
	effort	W	265	183	416	242	31%	42%
	months	SEESAW	21	15	13	8.6	27%	33%
	months	W	22	20	15	11.1	5%	24%
			NASA93ii Ground					
	defects	SEESAW	2006	688	4254	2203	66%	48%
	defects	W	2007	933	3763	1121	54%	70%
	effort	SEESAW	240	95	390	166	61%	57%
	effort	W	177	81	361	156	54%	57%
	months	SEESAW	22	16	15	8.8	28%	41%
	months	W	21	17	14	6.2	19%	55%
			NASA93ii OSP					
*	defects	W	1586	767	3557	1741	52%	51%
	defects	SEESAW	1265	1696	3722	3077	-34%	17%
*	effort	W	210	99	557	179	53%	68%
	effort	SEESAW	150	174	411	372	-16%	10%
*	months	W	21	15	15	9.0	28%	39%
	months	SEESAW	21	21	15	12	-2%	21%
			NASA93ii OSP2					
*	defects	W	2077	744	4222	1356	64%	68%
	defects	SEESAW	2042	1172	4369	3127	43%	28%
*	effort	W	239	79	465	145	67%	69%
	effort	SEESAW	210	118	514	275	44%	46%
	months	W	21	15	17	6.8	31%	60%
	months	SEESAW	21	16	17	11	25%	36%

Figure 5.5: Changes in median and spread for the NASA93 dataset.

The gray cells in Figure 5.6 show optimization *failures*; i.e. a zero or negative improvement. $\mathcal{W}2$ failed less than SEESAW (had fewer gray cells). $\mathcal{W}2$ showed $3/24$ and $7/24$ failures for medians and spreads (respectively) while SEESAW showed $13/24$ and $7/24$ failures for medians and spreads (respectively). One of SEESAW's failures was particularly dramatic: witness the increase from 98 effort months to 447 effort months in the OSP2 effort results. We conjecture that SEESAW's greater failures in median reduction are due to the over-fitting problem discussed in §2.4. SEESAW's model-based methods are free to sample increasingly narrow segments of the

COC81ii Flight								
	defects	W	1529	1265	1867	2369	17%	-27%
	defects	SEESAW	1487	1629	2054	1965	-9%	4%
	effort	W	86	81	181	200	6%	-11%
	effort	SEESAW	89	106	246	237	-19%	4%
*	months	W	18	16	6.5	10	11%	-49%
	months	SEESAW	18	20	10	8.9	-8%	8%
COC81ii Ground								
	defects	W	1541	1248	1902	2102	19%	-11%
	defects	SEESAW	1650	1496	2445	2499	9%	-2%
	effort	W	98	65	199	223	33%	-12%
	effort	SEESAW	106	122	383	372	-15%	3%
*	months	W	18	15	9.2	10	17%	-7%
	months	SEESAW	19	19	10	10	0%	-5%
COC81ii OSP								
*	defects	W	1496	1068	1787	2054	29%	-15%
	defects	SEESAW	1496	1765	2233	2233	-18%	0%
	effort	SEESAW	93	83	332	200	11%	40%
	effort	W	88	93	209	205	-5%	2%
*	months	W	19	14	9.0	8.9	22%	1%
	months	SEESAW	19	19	9.4	10	-3%	-4%
COC81ii OSP2								
*	defects	W	1850	1802	2697	2405	3%	11%
	defects	SEESAW	1473	2269	1769	2061	-54%	-17%
*	effort	W	122	130	431	356	-7%	17%
	effort	SEESAW	98	447	289	288	-356%	0%
	months	SEESAW	19	19	7.9	8.6	-3%	-9%
	months	W	20	21	11	10	-4%	10%

Figure 5.6: Changes in median and spread for the COC81 dataset.

internal state space of a model ("flying in", as it were, into small cracks between the training data). If that sampling is taken to extreme, and the model-based methods offer recommendations that cover a tiny part of the state space, and if the test data does not fall into that tiny region, then the model-based recommendations will fail.

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

In summary:

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

Chapter 6

Discussion

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

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

In that circumstance, discussions about how to best change a project can use results borrowed from other sites. For example, Figure 6.1 show's Boehm et al.'s [10] analysis of the effects of changing some project attribute from its minimum to maximum value. Based on data from a regression analysis of 161 projects, this figure comments that changing (e.g.) personnel/team capability can alter the effort to build software by up to 350%. Using this data, the effects of various changes can be investigated using Boehm's *delta* analysis technique [8]:

- An old project with known efforts is used as a baseline. A change to a project is described as a new project, expressed in terms of deltas to the variables of Figure 6.1.
- The new estimate is then the product of the baseline times the effort multiplier deltas.
- The "best" changes to a project are those that are simplest to implement and have most positive impact on the effort (ideally, reduces it).

Column two of Figure 6.1 lets us compare context-independent reasoning (e.g. delta analysis) vs. context-dependent reasoning (e.g. $\mathcal{W}2$). Note how that only a third of the Figure 6.1 attributes appear in the "best" treatments of Figure 4.8. Curiously, the two attributes with greatest impact (personnel/team capability and product complexity) are absent from Figure 4.8.

Why is $\mathcal{W}2$ ignoring an attribute with such a large impact (350%)? To answer that question, we have go to the context-dependent particulars. Recall from Figure 5.1 that in OSP2, product complexity is fixed at $cplx = 4$ and personnel/team capability is fixed at $pcap = 3$. $\mathcal{W}2$ does not recommend treatments for things that cannot change. Hence, $cplx$ and $pcap$ are absent from the OSP2 results of Figure 4.8. Similarly, OSP allows only $pcap = 3$ so this attribute is also absent.

The same reasoning does not explain the other absent attributes. To understand these, we must look at the data. OSP sets $cplx \in \{5,6\}$. This attribute is absent in the treatments since there is insufficient support in NASA93ii to justify their inclusion (there only five $cplx = 5$ examples in NASA93ii and no examples of $cplx = 6$). Similar explanations can explain all the remaining absences. Examples such as these show how $\mathcal{W}2$ can provide recommendations that may go against common expert advice. This lack of a defined relationship between data attributes underscores the need for careful query construction. For example, if a query contains conflicting attributes, $\mathcal{W}2$ maintains no internal inconsistency check. Model-based approaches such as S-COST [9] can provide this sanity check, but incur the costs associated with model-based methods discussed above (ontology restrictions, untamed internal model variance, etc).

In summary, when data is absent, managers can debate changes to projects by reusing data like Figure 6.1. However, the conclusions reached from a context-independent reasoning (like delta analysis) can be made more specific with local information about the kinds of projects seen in the local environment and the kinds of changes the local managers are willing to accept. Therefore, where possible, we recommend collecting local data and analyzing it with $\mathcal{W}2$.

id	appears in Figure 4.8 as	features	relative weight
1		Personnel/team capability	3.53
2		Product complexity	2.38
3	time	Time constraint	1.63
4	rely	Required software reliability	1.54
5		Multi-site development	1.53
6		Doc. match to life cycle	1.52
7		Personnel continuity	1.51
8	apex	Applications experience	1.51
9		Use of software tools	1.50
10		Platform volatility	1.49
11	stor	Storage constraint	1.46
12	pmat	Process maturity	1.43
13	ltex	Language & tools experience	1.43
14	sced	Required dev. schedule	1.43
15		Data base size	1.42
16		Platform experience	1.40
17		Arch. & risk resolution	1.39
18		Precedentedness	1.33
19		Developed for reuse	1.31
20		Team cohesion	1.29
21		Development mode	1.32
22		Development flexibility	1.26

Figure 6.1: Relative effects on development effort. From [8].

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 [64]. 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 [67] 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.

6.3 Scope of the Study

This study use conveniently available datasets in the PROMISE repository, the result applies within the same context of the datasets. In addition, our evaluation compares the performance of different methods across a finite number of problems, so it cannot be used to predict which method will be superior to others for some future, as yet unseen, problem. In fact, no method has been found so far that is universally superior to others in all problems; indeed, the "no-free-lunch theorem" [75] suggests that such an universal best method for all problems can never exist. In practice, for a given new learning problem, various methods need to be empirically evaluated to find the best ones, such as the ones carried out in the study.

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

That being said, we note that publications from other research communities assess their models in the same manner as this paper: see the *effort estimation* [30, 36, 37, 40, 72] and *defect prediction* [28,52,58,71] literature. Ideally, researchers in effort estimation, defect prediction, or learning changes to software projects should apply their recommendations to live projects. However, hold-out tests are widely used due to the tremendous practical difficulties associated with performing such tests on multiple software projects. At the very least, studies like this paper are required to prune the space of methods to be laboriously tested on new, real-world, projects.

Chapter 7

Conclusion

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

- Optimization - §4.2 shows that $\mathcal{W}2$ runs faster than our initial $\mathcal{W}$ algorithm. $\mathcal{W}2$ can be applied to large datasets quickly (Figure 4.3), and without sacrificing performance (Figure 4.4).
- Explanation - $\mathcal{W}$ and $\mathcal{W}2$ are simple implementations of Contrast Set Learning (§3.2), an easy to explain, intuitive learning process.
- Certification - $\mathcal{W}2$ performs as well or better than a multitude of software quality optimization techniques (§5.4). Including parametric modeling techniques (SEESAW, §5.5), drastic project changes (§4.5), and multiple data sets based on various case descriptions (§4.3).
- Application - $\mathcal{W}2$ can be applied to find locally learned recommendations that offer potentially unconsidered avenues for software quality optimization (§3.3).

In comparing the merits of a model-lite, case-based approach to a parametric one, advocates of reconstructive memory such as Barlett [6], Kolodner [33], or Shank [64] 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. [4]), 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. 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 2.2). 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 3.1. $\mathcal{W}2$ extends standard CBR by learning an adaption of q, called q', that retrieves *better* quality examples. Based on the analysis of [31] and this paper, we recommend $\mathcal{W}2$ on several grounds:

- $\mathcal{W}2$ finds similar, or better, results than SEESAW (see §5.5).
- $\mathcal{W}2$ 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 [31].

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. 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 [45] 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.

Appendix A

$\mathcal{W}$2 Source Code

A.1 w.sh

```
if [ $Verbose -ne 0 ]
then
        echo "This is \"W\" (the decider) [v2.0]"
        echo "(c) 2009, GPL3.1 Adam Brady, Tim Menzies, Jackie Keung"
        date
        echo ""
        echo "historical data : $1"
        echo "new project(s)  : $2"
fi

Discretize=${Discretize:-0}
BinNames=${BinNames:-0}
MinOverlap=${MinOverlap:-0.75}
Tmp=$HOME/tmp
Samples=${Samples:-50}
K1=${K1:-5}
K2=${K2:-15}
Tests=${Tests:-0.33}
Seed=${Seed:-$RANDOM}
Nomograms=${Nomograms:-0}
AutoStop=${AutoStop:-1}
Log=${Log:-0}
```

```
Verbose=${Verbose:-1}
RankedOverride=${RankedOverride:-0}
Note=${Note:-0}
Class=${Class:-0}
KNN=${KNN:-0}

if [ $Discretize -eq 0 ]
then
        cat datasets/$1.dat projects/$2 > $Tmp/w-data.tmp
else
        cat datasets/$1.dat projects/$2 |
        gawk -f scripts/discretize.awk -v BinNames=$BinNames \
        -v Discretize=$Discretize > $Tmp/w-data.tmp
fi

cat $Tmp/w-data.tmp |
        pgawk \
        --dump-variables=$Tmp/vars10  \
        --profile=$Tmp/prof10          \
        -f w.awk -f apply.awk -f contrast.awk\
        -f neighbors.awk -f projects.awk -f util.awk\
        --source 'END {
                main()
        }' Tests=${Tests} Samples=$Samples K1=${K1} K2=${K2} Seed=${Seed} \
                Nomograms=$Nomograms AutoStop=$AutoStop ProjName=$2 Log=$Log \
                MinOverlap=$MinOverlap RankedOverride=$RankedOverride \
                Verbose=$Verbose Note=$Note Class=$Class \
                SkipRelevancy=$SkipRelevancy KNN=$KNN
```

A.2 w.awk

```
BEGIN { # command-line options
                Samples = 20
                K1          = 5
                K2          = 15
                Seed    = 1
                Tests   = 0.33
                AutoStop = 1
                MinOverlap = 0.75
```

```
                RankedOverride = 0
                Verbose = 1
                Note = ""
                Class = ""
                KNN = ""
}
BEGIN { # internal options
                OFS=","
                IGNORECASE=1
                Inf = 10^32
                _ = SUBSEP
                CONVFMT="%.8g"
                OFMT="%.8g"
}

##################################################################
# main program

function main() {
        worker(Samples,K1,K2)
}
function worker(samples, k1,k2,        rankeds, ranked) {
    if (Verbose) {
        print "samples          : " samples
        print "k1               : " k1
        print "k2               : " k2
        print "%test            : " Tests*100
        print "Contrast Method : " (Nomograms ? "Nomograms" : "BSquared")
        print "Case Relevancy  : " (MinOverlap ? MinOverlap*100"% Overlap" : "Stoicastic Samples"
            )
        print "Logging          : " (Log ? "On ("Log")" : "Off")
        print ""
        print (RankedOverride ? "Overriding training recomendations using "RankedOVerride : "
            Training results on " Train[0] " historical examples (what looks useful):")
    }

    rankeds = (RankedOverride ? injectRanked(RankedOverride, ranked) : train(samples, k1, k2,
        ranked))
```

```
    printf (Verbose ? "Test␣results␣on␣" Test[0] "␣new␣projects␣(applying␣the␣training␣results␣to
        ␣new␣data):\n" : "")
    test(samples,k1,k2,rankeds,ranked)
}
function train(samples,k1,k2,ranked,     projects,neighbors,memos,best,rest,knearest,rankeds) {

    if (!KNN) {
        getRelevant(Train,k1+k2,relevant)
        bestRest(relevant,k1,                     best,rest)       # divide knearest into best
            /worst
    }
    else {
        getProjects(MinOverlap,Train,samples,     projects)        # get example1 projects
        neighbors(samples,projects,Train[0],Train,  neighbors,memos) # distances from example1
            to Train cases
        knn(k1+k2,samples,neighbors,memos,        knearest)        # knearest Train instance
            row numbers to example1 p
        bestRest(knearest,k1,                     best,rest)       # divide knearest into best
            /worst
    }

    rankeds = rank(k1,k2,best,rest,            ranked)         # contrast set between best/
        worst
    return rankeds
}

function test(samples,k1,k2,rankeds,ranked,    \
              i,projects,neighbors,memos,knearest,     \
              m,n,sorted,kloc,row,col,data,rowKlasses) {

    if (!KNN) {
        getRelevant(Test,k1+k2-1,knearest)
    }
    else {
        getProjects(MinOverlap,Test,samples,      projects)        # get example2 projects
        neighbors(samples,projects,Test[0],Test,   neighbors,memos) # distances example2 to
            Test set
        knn(k1+k2-1,samples,neighbors,memos,        knearest)        # knearest Test instances
             row numbers to example2 projects
```

```
    }

    for(row=1;row<=Test[0];row++) #Re-align indexes for knn data
        if (row in knearest) {
            data[0]++

            for(col=1;col<=Cols;col++)
                data[data[0],col]=Test[row,col]          # convert row numbers to their data
                    rows
        }
    apply(ranked,data)
}
#################################################################
# read in data

                { gsub(/%.*/,"") }
/^[     \t]$/       { next }
/^@project/     { In = 0 }
In              { rand() <= Tests ? cells(Test,Cols) : cells(Train,Cols) }
/^@relation/    { Relation=$2 }
/^@attribute/   { def($2) }
/^@class/       { defclass($2) }
/^@data/        { In = 1; inits(Cols) }
/^@/            { next }

function inits(cols,  i) {
    srand(Seed ? Seed : 1)
    for(i=1;i<=cols;i++) { Train["max",i]= -1*Inf; Train["min",i]=Inf }
    for(i=1;i<=cols;i++) { Test[ "max",i]= -1*Inf; Test[ "min",i]=Inf }
}
function def(name,  a,i,goalp) {
    goalp  = sub(/?/,"",name)
    if (name in Name)  {
        a = Name[name]
    } else {
        a = Name[name] = ++Cols
        Eman[Cols]=name
    }
    if (Train["range",a,0]) {
```

```
        clearStack(Train, "range" _ a)
        clearStack(Test, "range" _ a)
    }

    for(i=3;i<=NF;i++) {
        Train["range",a, ++Train["range",a,0]] = $i
        Test[ "range",a, ++Test[ "range",a,0]] = $i
    }
    if (goalp) Goal[a]=1
}
function defclass(name) {
    if (name in Name)  {
        a = Name[name]
    } else {
        a = Name[name] = ++Cols
        Eman[Cols]=name
    }
    if (Train["range",a,0]) {
        clearStack(Train, "range" _ a)
        clearStack(Test, "range" _ a)
    }

    for(i=3;i<=NF;i++) {
        Train["range",a, ++Train["range",a,0]] = $i
        Test[ "range",a, ++Test[ "range",a,0]] = $i
    }
    if (Class) {
        if (Class ~ name) {
            Klasses[name] = Name[name]
            NumKlasses++
        }
    }
    else {
        Klasses[name] = Name[name]
        NumKlasses++
    }
}
function clearStack(a, key,    i, max) {
    if (max = a[ key _   0 ])
```

```
        for(i=1;i<=max;i++)
            delete a[key _ i]
    a[key _ 0] = 0
}
function cells(data,cols,     col) {
    data[0]++
    for(col=1;col<=cols;col++)  {
        data[data[0],col] = $col
        data["max",col]   = max(data["max",col],$col)
        data["min",col]   = min(data["min",col],$col)
    }
}
```

A.3 apply.awk

```
##########################################################################
# select and report subset of relevant rows that satisfy constraints 1..n

function apply(treatments,data,     baseEstimateData,finalEstimateData,constraints,t,optimal,
    filteredData) {

    #Save the original estimate data
    for (r=1; r<=data[0]; r++) {
        for (c=1; c<=Cols; c++) {
            baseEstimateData[r,c] = data[r,c]
        }
    }
    baseEstimateData[0] = data[0]

    getClassRanges(baseEstimateData,            classRanges)
    scoreData(baseEstimateData, classRanges,     baseRowToScore)

    baseMedian = findMedian(baseRowToScore)
    baseSpread = findSpread(baseRowToScore)

    if (Verbose) {print "\n\n------------------Queries------------------"}

    if (Verbose) {describeData("Query #0", baseEstimateData, baseRowToScore)}
```

```
optimal=0
t=1
previousMedian = baseMedian
previousSpread = baseSpread
while(!optimal) {
    #          ------input------          ----output----
    addConstraint(treatments[t],          constraints)
    filter(data, constraints,             filteredData)

    split("",scores,"")
    scoreData(filteredData, classRanges,  scores)

    filteredMedian = findMedian(scores)
    filteredSpread = findSpread(scores)

    #Stopping Rules
    if (!treatments[t]) {
        optimal=1
        printf (Verbose ? "STOPPING. No more treatments left:\n" : "")
    }
    if (filteredData[0] < 3) {
        optimal=1
        printf (Verbose ? "STOPPING. Next query too small:\n" : "" )
    }
    if (filteredMedian >= previousMedian && filteredSpread >= previousSpread) {
        optimal=1
        printf (Verbose ? "STOPPING. Next query shows no improvement:\n" : "" )
    }
    if (t == 1) #first treatment always works
        optimal=0

    previousMedian = filteredMedian
    previousSpread = filteredSpread

    if (!optimal) {
        split("",finalEstimateData,"")
        split("",finalScores,"")
        for (r=1; r<=filteredData[0]; r++) {
            for (c=1; c<=Cols; c++) {
```

```
                finalEstimateData[r SUBSEP c] = filteredData[r SUBSEP c]
            }
            finalEstimateData[0] = filteredData[0]
            finalScores[r] = scores[r]

        }
    }
    if (Verbose) {describeData("Query␣#"t, filteredData, scores) }
    t++
}

if (Verbose) {print "———————————Results———————————"}

if (Verbose) {describeData("Baseline", baseEstimateData, baseRowToScore) }

for (col in constraints) {
    split(constraints[col],tmp,SUBSEP)
    for (val in tmp)
        recommendation = recommendation Eman[col]"="tmp[val]"␣"
}

if (Verbose) {describeData("Final", finalEstimateData, finalScores)}

baseScoreMedian = findMedian(baseRowToScore)
finalScoreMedian = findMedian(finalScores)

baseScoreSpread = findSpread(baseRowToScore)
finalScoreSpread = findSpread(finalScores)

if (finalScoreMedian == 0)
    scoreMedianReduction = 0
else
    scoreMedianReduction = 100 * (baseScoreMedian - finalScoreMedian) / baseScoreMedian

if (finalScoreSpread == 0)
    scoreSpreadReduction = 0
else
    scoreSpreadReduction = 100 * (baseScoreSpread - finalScoreSpread) / baseScoreSpread
```

```
outStr = "   score median: "sprintf("%4.0f",scoreMedianReduction)"%"
outStr = outStr"\n   score spread: "sprintf("%4.0f",scoreSpreadReduction)"%"

RankedOverride=""

if (Log) {
    if (NumKlasses > 1) {
    printf "score.ASIS."(Note ? Note : Relation)"."ProjName""(RankedOverride ? "."
        RankedOverride : "")"," >> Log
    printf arr2str(baseRowToScore) >> Log
    printf "\nscore.TOBE."(Note ? Note : Relation)"."ProjName""(RankedOverride ? "."
        RankedOverride : "")"," >> Log
    printf arr2str(finalScores) >> Log
    printf "\n" >> Log
    }
}

for(key in Klasses) {
    split("",tmpBase,"")
    split("",tmpFinal,"")
    for(r=1; r<=baseEstimateData[0]; r++) {
        tmpBase[r] = baseEstimateData[r,Klasses[key]]
    }
    for(r=1; r<=finalEstimateData[0]; r++) {
        tmpFinal[r] = finalEstimateData[r,Klasses[key]]
    }
    baseMedian = findMedian(tmpBase)
    finalMedian = findMedian(tmpFinal)

    baseSpread = findSpread(tmpBase)
    finalSpread = findSpread(tmpFinal)

    if (finalMedian == 0)
        medianReduction = 0
    else
        medianReduction = 100 * (baseMedian - finalMedian) / baseMedian

    if (finalSpread == 0)
        spreadReduction = 0
```

```
        else
            spreadReduction = 100 * (baseSpread - finalSpread) / baseSpread

        outStr = outStr"\n"sprintf("%8s",key)" median: "sprintf("%4.0f",medianReduction)"%"
        outStr = outStr"\n"sprintf("%8s",key)" spread: "sprintf("%4.0f",spreadReduction)"%"

        if (Log) {
            printf key".ASIS."(Note ? Note : Relation)"."ProjName""(RankedOverride ? "."
                RankedOverride : "")"," >> Log
            printf arr2str(tmpBase) >> Log
            printf "\n"key".TOBE."(Note ? Note : Relation)"."ProjName""(RankedOverride ? "."
                RankedOverride : "")"," >> Log
            printf arr2str(tmpFinal) >> Log
            printf "\n" >> Log
        }
    }
    if (Verbose) {
        print ""
        print "----------Reduction Summary----------"
        print outStr
    }
        print recommendation
    if (!Verbose) {printf "."}
}

function sortData(data, scores) {

    for (row in scores) {
        copy[row] = 0.000001 * rand() + scores[row]
        serocs[copy[row]] = row
    }

    data[0] = newdata[0] = asort(copy)

    for (row=1; row<=newdata[0]; row++) {
        for (c=1; c<=Cols; c++)
            newdata[row SUBSEP c] = data[serocs[copy[row]] SUBSEP c]
    }
```

```
    for (row=1; row<=newdata[0]; row++) {
        for (c=1; c<=Cols; c++)
            data[row SUBSEP c] = newdata[row SUBSEP c]
    }
}

function getClassRanges(data,  ranges,     class,min,max,colnum,row) {
    for (class in Klasses) {
        min = Inf
        max = -1*Inf
        colnum = Klasses[class]
        for (row=1; row<=data[0]; row++) {
            if (data[row SUBSEP colnum] > max)
                max = data[row SUBSEP colnum]
            if (data[row SUBSEP colnum] < min)
                min = data[row SUBSEP colnum]
        }
        ranges[colnum SUBSEP "min"]=min
        ranges[colnum SUBSEP "max"]=max
    }
}

function scoreData(data, ranges,  scores,      numClasses,key,r,score,class,colnum,min,max) {
    if (!data[0])
        "Can't score data with no defined size"

    numClasses=0
    for(key in Klasses)
        numClasses++

    for (r=1; r<=data[0]; r++) {
        score=0
        for (class in Klasses) {
            colnum = Klasses[class]
            min    = ranges[colnum SUBSEP "min"]
            max    = ranges[colnum SUBSEP "max"]

            if (max - min == 0)
                score += 0
```

```
            else
                score += (data[r SUBSEP colnum] - min) / (max - min)
        }
        scores[r] = score/numClasses #Normalize score to 1.0
    }
}

function describeData(name, data, scores) {
    strdesc = ""
    print name" (size: "data[0]")"

    asort(scores, scoresCopy)
    print "\tScore-Median: "findMedian(scoresCopy)
    print "\tScore-Spread: "findSpread(scoresCopy)

    strdesc = findMedian(scoresCopy)","findSpread(scoresCopy)

    printf "\tScores: "
    for (r=1; r<=data[0]; r++)
        printf "%.3f ",scoresCopy[r]
    print ""

    for (class in Klasses) {
        split("",tmp,"")
        printf "\t"class": "
        t=0
        for (r=1; r<= data[0]; r++)
            tmp[++t] = data[r SUBSEP 23]#"("data[r SUBSEP Klasses[class]]")"
        asort(tmp)
        for (r=1; r<= data[0]; r++)
            printf tmp[r]" "
        print ""
        print "\t"class"-median: "findMedian(tmp)
        print "\t"class"-spread: "findSpread(tmp)

        strdesc = strdesc","findMedian(tmp)","findSpread(tmp)
    }
    print ""
```

```
    return strdesc
}

#Add a treatment to the list of constraints
function addConstraint(treatment, constraints) {
    split(treatment,tmp,SUBSEP)
    attr = tmp[1]
    val  = tmp[2]

    if(!Seen[attr SUBSEP val]) {
        if(constraints[attr]) #Have we already constrained this attribute?
            constraints[attr] = constraints[attr] SUBSEP val #if so, extend its range
        else
            constraints[attr] = val
    }

    Seen[attr SUBSEP val]=1
}


function filter(data, constraints,   filteredData,     tmp,passesNeeded) {
    split("",filteredData,"")
    passesNeeded=0
    for (key in constraints)
        passesNeeded++

    for(row=1; row<=data[0]; row++) {
        #Row passes if all attributes match constraint values
        #Disjunctions form if multiple constraint values for a single attribute exist
        passes=0
        for(col in constraints) {
            success=0
            split(constraints[col], possibleValues, SUBSEP)
            for (value in possibleValues) {
                if (possibleValues[value] == data[row,col]) {
                    success=1
                }
            }
```

```
        if(success) { #Guarantees we can't somehow match multiple times in a range for a
            single attr
            passes++
        }
    }

    if(passes == passesNeeded) { #All constraints matched
        filteredData[0]++ #Ensure monotonic increasing order in filtered set indexes
        for(c = 1; c <= Cols; c++) {
            filteredData[filteredData[0],c] = data[row,c] #Add to filtered set
        }
    }
  }
}
```

A.4 contrast.awk

```
#################################################################
# divide the k-th nearest historial projects into best (lowest)
# estiamted and the rest. collect frequency counts for best and rest.
# rank attribute ranges by how common they are in best and how
# rare they are in rest

function bestWorst(rows,border,best,rest,     cutoffBest,cutoffWorst,n,scores,row,k,r,rowKlasses)
  {
    for(row in rows) {
        scores[++n] = scoreRow(row, Train, Klasses)
    }

    n = asort(scores)

    cutoffBest = scores[border]
    cutoffWorst = scores[n-border]

    for(row in rows) {
        if ( scoreRow(row, Train, Klasses) <= cutoffBest ) {
            count(row,best,rows[row])
            print("BEST")
        }
```

```
            if ( scoreRow(row, Train, Klasses) > cutoffWorst ) {
                count(row,rest,rows[row])
                print("WORST")
            }
        }
}

function bestRest(rows,border,best,rest,     enough,n,scores,row,k,r,rowKlasses) {

        for(row in rows) {
                scores[++n] = scoreRow(row, Train, Klasses)
        }

        asort(scores)
        enough = scores[border]
        for(row in rows) {
                if ( scoreRow(row, Train, Klasses) <= enough ) {
                        count(row,best,rows[row])
                }
                else {
                        count(row,rest,rows[row])
                }
        }
}
function count(row,f,n,     col,c) {
    f[0]++
    for(col in Goal) {
        f[col,Train[row,col]] += n
    }
}
function rank(k1,k2,best,rest,ranked,    \
                range,bests,rests,i,b,r,score,scores,sorted,memo,max) {
        bests = best[0]
        rests = rest[0]
        for(i in best)  {
                if (i != 0) {
                        b          = best[i] / bests
                        r          = rest[i] / rests
                        score=as100((b^2)/(b+r))
```

```
                    scores[i]      = score
                    memo[score] = i
                }
        }
        max = asort(scores,sorted)
        showRanks(memo,sorted,max)
        for(i=max; i>=1;i--)  # highest score must be forst
                ranked[max-i+1] = memo[sorted[i]]

        return max
}
function showRanks (memo,sorted,max,   i, range,tmp,com) {
    if (Verbose) {
        com="sort -r -n | cat -n"
        print "#n\tscore\trange"
        print "------\t-----\t---------------"
        for(i=max; i>=1;i--) { # highest score must be forst
                range = memo[sorted[i]]
                split(range,tmp,_)
                print sprintf("%5.2f",sorted[i]) "\t" Eman[tmp[1]] " = " tmp[2]   | com
        }
        close(com)
        print ""
    }
}
```

A.5 discretize.awk

```
BEGIN{
    NumBins= (Discretize ? Discretize : 2)
    BinNames = (BinNames ? BinNames : 0)
    NoProj = (NoProj ? NoProj : 0)
}

                {gsub(/%.*/,"")}
/^[ \t]*$/     { next }
/@relation/    { print $0}#"-"NumBins"Bins" }
/@attribute/ { Proj ? printProj() : initCol() }
/@class/       { initClass() }
```

```
/@project/    { print $0; Proj=1; In=0 }
In            { printData() }
/^@data/      { print $0; mapBins(); In=1 }
/^@/          { next }

function initCol(   n,i, seen,binned,name) {
        Columns[++Cols] = $2
        name = $2
        sub(/?/,"",name)
        ColName2Indx[name] = Cols
        for (i = 1; i <= NF-2; i++) {
            ColValues[Cols,i] = $(i+2) #@attr name 1stval 2ndval 3rdval...
        }
        mapBins(ColValues, Cols, NF-2, BinMap)
        printf $1" "$2
        for (i = 1; i <= NF-2; i++) {
            binned = BinMap[Cols,$(i+2)]
            if (!seen[binned]) {
                seen[binned] = 1
                printf " "BinMap[Cols,$(i+2)]
            }
        }
        print ""
}

function initClass() {
    Columns[++Cols] = $2
    Class[Cols] = 1
    printf $1" "$2
    for (i = 3; i <= NF; i++) {
        printf " "$i
    }
    print ""
}

#BinMap[columnNumber, value] = bin
function mapBins(ColValues, colIndx, numValues, BinMap,      tmp,i,c) {
    for (i = 1; i <= numValues; i++) {
        val = ColValues[colIndx,i]
```

```
        binWidth = numValues/NumBins
        bin = int((i-1)/binWidth)+1

        if (BinNames)
            BinMap[colIndx,val] = "B"(bin)
        else
            BinMap[colIndx,val] = ColValues[colIndx,int(numValues*((bin-1)/NumBins))+1]

        if (BinNames && NumBins=5)
            BinMap[colIndx,val] = rangeName(bin)
    }
}

function rangeName(bin,      name) {
    name = ""
    if (bin == 1)
        name = "vl"
    if (bin == 2)
        name = "lo"
    if (bin == 3)
        name = "md"
    if (bin == 4)
        name = "hi"
    if (bin == 5)
        name = "vh"
    return name
}

function printData() {
    for (c = 1; c <= Cols; c++) {
        if (Class[c])
            printf $c" "
        else
            printf BinMap[c,$c]" "
    }
    print ""
}

function printProj(     i,seen) {
```

```
    printf $1" "$2
    for (i=3; i<=NF; i++) {
        name = $2
        sub(/?/,"",name)

        binned = BinMap[ColName2Indx[name],$i]
        if (!seen[binned]) {
            seen[binned] = 1
            printf " "binned

        }
    }
    print ""
}
```

A.6 neighbors.awk

```
#################################################################
# find the k-th nearest historial projects near the generated projects

function euclidean(row1,row2,data1,data2,     n,col,d,d1,d2,key,ignorep,i) {
        split("",ignorep,"")
        for (key in Klasses)
                ignorep[Klasses[key]] = 1
        for(col=1;col<=Cols;col++)
            if (!(col in ignorep)) {
                        d1 = normalize(data1,col,data1[row1,col])
                        d2 = normalize(data2,col,data2[row2,col])
                        d += abs(d1 - d2)^2
                        n++
        }
        return  sqrt(d)/sqrt(n)
}
function distance(row1,row2,data1,data2,memo,   d) {
        d = as100(euclidean(row1,row2,data1,data2))
        memo[-1 * d] = row1  # d started at row1
        memo[d]      = row2  # d ended at row2
        return d
}
```

```
function normalize(data,col,n,      min,max,d) {
        min = data["min",col]
        max = data["max",col]
        d = min == max ? 1 : (n - min) /(max - min)
        return d
}
function neighbors(news,new,olds,old,neighbor,memo,   o,n) {
        for(n=1;n<=news;n++)
                for(o=1;o<=olds;o++)
                        push2(distance(n,o,new,old,memo),neighbor,n)
}
function knn(k,news,neighbor,memo,ks,     dist,n,most,i,d,sorted) {
        for(n=1;n<=news;n++) {
                most = neighbor[n,0]
                for(i = 1;i <=  most;i++)
                        dist[++d] = neighbor[n,i]
        }
        knnDebug(dist,memo)
        asort(dist,sorted)
        for(i=1;i<=d;i++)  {
                n  = memo[sorted[i]]
                if ( ++ks[n]==1 ) k--
                if (k == 0)  return i
    }
        return k
}
```

A.7 projects.awk

```
##################################################################
# generate projects

function getRelevant(data, k, relevant,      overlap) {
    for(row=1; row<=data[0]; row++) {
        for (col=1; col<=Cols; col++) {
            if (projectContains(data[row,col],col,data)) {
                overlap[row]++
            }
        }
```

```
        overlap[row] += 0.01 * rand()
        rowLookup[overlap[row]] = row
    }

    m = asort(overlap)

    for (i=m; i>=m-k; i--) {
        relevant[rowLookup[overlap[i]]] = 1
    }
}

function getProjects(minOverlap, data, samples,     projects) {
    if(minOverlap) {
        rejectProjects(minOverlap,data,samples,     projects)
        printf (Verbose ? "Found "projects[0]" projects out of "data[0]" with "100*minOverlap"% 
            attribute overlap (values contained within project ranges)\n" : "")
    }
    else {
        generateProjects(data,samples,     projects)
        printf (Verbose ? "Generated "samples" samples from project ranges\n" : "")
    }
}

function rejectProjects(minOverlap, data, news, new,     row,col,pass) {
    for(row=1; row<=data[0]; row++) {
        fails = 0

        for(col=1; col<=Cols; col++) {
            if (!projectContains(data[row,col],col,data)) {
                fails++
            }
        }

        if (fails < (1-minOverlap) * Cols) {
            new[0]++
            for (col=1; col<=Cols; col++)
                new[new[0],col]=data[row,col]
        }
```

```
    }

    #Find mins and maxes of accepted projects
    for(col=1; col<=Cols; col++) {
        new["max",col]= -1*Inf
        new["min",col]= Inf
        for(row=1;row<=new[0]; row++)  {
            new["max",col] = max(new["max",col], new[row,col])
            new["min",col] = min(new["min",col], new[row,col])
        }
    }
}

function projectContains(value,col,data,      numValues,found,i) {
    numValues = data["range",col,0]

    found = 0
    for (i=1; i<=numValues; i++) {
        if (value == data["range",col,i])
            found = 1
    }

    return found
}

function generateProjects(data,news,new,   row,col,v) {
        new[0]=news
        for(col=1;col<=Cols;col++)  {
                new["max",col]= -1*Inf
                new["min",col]= Inf
                for(row=1;row<=news;row++)  {
                        v = projectValue(data,col)
                        new[row,col]    = v
                        new["max",col] = max(new["max",col],v)
                        new["min",col] = min(new["min",col],v)
                }
        }
}
function projectValue(data,a,    max,one) {
```

```
        max = data["range",a,0]
        one = int(rand() * max) + 1
        return data["range",a,one]
}
```

A.8 util.awk

```
##################################################################
# mann-whitney tests

function mwRank(data0,ranks,      data,starter,n,old,start,skipping,sum,i,j,r) {
    starter="someCraZYsymBOL";
    n     = asort(data0,data)
    old   = starter
    start = 1;
    for(i=1;i<=n;i++) {
        skipping = (old == starter) || (data[i] == old);
        if (skipping) {
            sum += i
        } else {
            r = sum/(i - start)
            for(j=start;j<i;j++)
                ranks[data[j]] = r;
            start = i;
            sum   = i;
        }
        old=data[i]
    }
    if (skipping)
        ranks[data[n]] = sum/(i - start)
    else
        if (! (data[n] in ranks))
            ranks[data[n]] = r+1
}

function mwu(x,pop1,pop2,up,critical,
            i,data,ranks,n,n1,sum1,ranks1,n2,sum2,ranks2,       \
            correction,meanU,sdU,z) {
```

```
    for(i in pop1) data[++n]=pop1[i]
    for(i in pop2) data[++n]=pop2[i]
    mwRank(data,ranks)
    for(i in pop1) { n1++; sum1 += ranks1[i] = ranks[pop1[i]] }
    for(i in pop2) { n2++; sum2 += ranks2[i] = ranks[pop2[i]] }

    meanU      = n1*(n1+n2+1)/2  # symmetric, so we just use pop1's z-value
    sdU        = (n1*n2*(n1+n2+1)/12)^0.5
    correction = sum1 > meanU ? -0.5 : 0.5
    z          = abs((sum1 - meanU + correction)/sdU)

    if (z >= 0 && z <= critical)
        return 0
    if (up) {
        return 1
    }
    else {
        return -1
    }
}
function criticalValue(conf) {
    conf = conf ? conf : 95
    if (conf==99) return 2.326
    if (conf==95) return 1.960
    if (conf==90) return 1.645
}


function s2a(s,a,    tmp,i,n) {
    n=split(s,tmp,/ /)
    for(i=1;i<n;i+=2)
            a[tmp[i]]=tmp[i+1]
}

function median(arrin,n,   low,a) {
    low = int(n/2);
    return oddp(n) ? a[low+1] : (a[low] + a[low+1])/2
}
```

```
function multiple(a,n,  i) { for (i in a) a[i] *= n }
#function abs(x)           { return x < 0 ? -1*x : x }
function oddp(n)           { return n % 2 }


##################################################################
# standard utils

function barph(str)       { print str >"/dev/stderr"; fflush("/dev/stderr") }
function push2(v,a,i)     { a[i,++a[i,0]] = v; return v }
function push(v,a)        { a[++a[0]] = v; return v }
function as100(n)         { return (n*100) + rand()/100 }
function abs(n)           { return n < 0 ? -1* n : n }
function max(n1,n2)       { return n1<n2 ? n2 : n1 }
function min(n1,n2)       { return n1<n2 ? n1 : n2 }
function no_(str)         { gsub(_,",",str); return str }
function round(i)         { return int(i+0.5) }

function copya(source,copy,     key) {
        split("",copy,"") #clear copy
        for (key in source) {
                copy[key] = source[key]
        }
}



#doesn't interpolate for even number  of values
#(if you want to get actual cases back)
function findAbsMedian(a,       n,floor,sorted) {
        n = asort(a,sorted)
        floor = int(n/2)
        if (n == 1)
                return sorted[1]
        else
                return sorted[floor]
}
```

```
#doesn't interpolate for even number of values
#(if you want to get actual cases back)
function findAbsSpread(a,  n,floor25,floor75,sorted) {
        n = asort(a,sorted)
        floor75 = sorted[int(3*n/4)+1]
        floor25 = sorted[int(n/4)+1]
        return floor75 - floor25
}

function findMedian(a,  n,floor,sorted) {
        n = asort(a,sorted)
        floor = int(n/2)
        if (n == 1)
                return sorted[1]
        else
                return oddp(n) ?  sorted[floor+1] : (sorted[floor] + sorted[floor+1])/2
}

function medianReduc(asis,tobe) {
    return 100 * (findMedian(asis) - findMedian(tobe)) / findMedian(asis)
}



function spreadReduc(asis,tobe) {
    return 100 * (findSpread(asis) - findSpread(tobe)) / findSpread(asis)
}

function findSpread(a, len,sorted) {
    len = asort(a, sorted)
    if (len == 2)
        return sorted[2] - sorted[1]

    if (len < 2)
        return 0

    return find75(a) - find25(a)
}

function findAbs25(a,   n,sorted) {
```

```
        n = asort(a,sorted)
        return sorted[int(n/4)+1]
}

function findAbs75(a,    n,sorted) {
        n = asort(a,sorted)
        return sorted[int(3*n/4)+1]
}

function find25(a,       n,floor25,sorted) {
        n = asort(a,sorted)
        n%4 == 1 ? floor25 = sorted[int(n/4)+1] : floor25 = (sorted[int(n/4)] + sorted[int(n/4)
            +1]) / 2
        return floor25
}

function find75(a,       n,floor75,sorted) {
        n = asort(a,sorted)
        n%4 == 1 ? floor75 = sorted[int(3*n/4)+1] : floor75 = (sorted[int(3*n/4)] + sorted[int(3*
            n/4)+1]) / 2
        return floor75
}

function arrMax(a,    n,sorted) {
    n = asort(a, sorted)
    return sorted[n]
}

function arrMin(a,    n,sorted) {
    n = asort(a, sorted)
    return sorted[1]
}

function findAvg(a,      n,sorted,sum,num,i) {
    n = asort(a,sorted)
    for (i=1; i<=n; i++) {
        sum += sorted[i]
        num ++
    }
```

```
    return sum / num
}

function findStdev(a,     n,sorted,sum,num,mean,i,sumsq) {
    n = asort(a,sorted)
    for (i=1; i<=n; i++) {
        sum += sorted[i]
        num ++
    }
    mean = sum / num

    for (i=1; i<=n; i++) {
        sumsq += (sorted[i] - mean) * (sorted[i] - mean)
    }

    return sqrt(sumsq / num)
}


function arr2str(a,sep,    s,n,tmp) {
    sep = (sep ? sep : ",")
    n = asort(a,tmp)
    for (i=1; i<=n; i++) {
        s = (s ? s sep a[i] : a[i])
    }
    return s
}

function saya(a,s,    b,c,m,n,key,val,i,j,tmp,sep) {
        print ""
        m     = asorti(a,b)
        for(i=1;i<=m;i++)   {
                key=b[i]
                val=a[b[i]]
                printf("%s",   sep s "["   )
                n=split(key,tmp,_)
                c = ""
                for(j=1;j<=n;j++)         {
                        printf("%s", c tmp[j]   )
```

```
                    c=","
                }
                if (val ~ _)
                        val = no_(val)
                printf("%s", "]=" val )
                sep="\n";
        };
        print ""
}

function scoreRow(row, data, classes,    maxes, mins,classCount,normClassVal,k,score,sumsqr,
    relyCol) {
    #normalize values
    for (k in classes) {
        maxes[k] = max(Test["max",classes[k]],Train["max",classes[k]])
        mins[k] = min(Test["min",classes[k]],Train["min",classes[k]])
        classCount++
        normClassVal[k] = ( data[row,classes[k]] - mins[k] ) / ( maxes[k]-mins[k] )
    }

    #Single-goal scoring (return goal value)
    if (classCount == 1) {
        score = data[row,classes[k]]
    }
    #Multi-goal scoring (collapse to a single value)
    else {

        #Normalized Euclidean
        if(ScoreMethod == 0) {
            sumsqr = 0
            for (k in normClassVal) {
                sumsqr += ( normClassVal[k] * normClassVal[k] )
            }
            score = sqrt(sumsqr)
        }

        #BFC (bias defects)
        if(ScoreMethod == 1) {
            relyCol = Name["rely"]
```

```
            if (!relyCol)
                print "ERROR: No rely column for BFC formula:" relyCol
            if (!("defects" in classes))
                print "No defects class for BFC formula"

            sumsqr = 0
            for (k in normClassVal) {
                if (k ~ "defects") {
                    print "relyval:" data[row,relyCol]
                    sumsqr += (normClassVal[k] * (1 + 1.8^(data[row,relyCol] - 3)))^2
                }
                else
                    sumsqr += normClassVal[k]^2
            }
            score = sqrt(sumsqr)
        }

    }
    return score
}

function injectRanked(rules, ranked,        r) {
    print "injecting"
    r=0
    while(getline < rules) {
        n = Name[$1]
        if (!($1 in Name))
            print "Error: Couldn't find "$1" in the dataset"
        else
            print "Added "$1","$2" successfully"
        ranked[++r] = n SUBSEP $2

    }
    return r
}
```

Appendix B

Example Dataset and Project Descriptions

B.1 NASA93 Project Descriptions

B.1.1 NASA Ground Software

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

B.1.2 NASA Flight Software

```
@project
@attribute ?rely 3 4 5
```

```
@attribute ?data 2 3
@attribute ?cplx 3 4 5 6
@attribute ?time 3 4
@attribute ?stor 3 4
@attribute ?acap 3 4 5
@attribute ?pcap 3 4 5
@attribute ?apex 2 3 4 5
@attribute ?plex 1 2 3 4
@attribute ?ltex 1 2 3 4
@attribute ?pmat 2 3
@attribute tool 2
@attribute sced 3
```

B.1.3 NASA Orbital Space Plane (OSP)

```
@project
@attribute ?pmat 1 2 3
@attribute rely 5
@attribute data 3
@attribute ?cplx 5 6
@attribute ?stor 3 4 5
@attribute pvol 2
@attribute ?acap 2 3
@attribute pcap 3
@attribute ?apex 2 3
@attribute plex 3
@attribute ?ltex 2 3 4
@attribute ?tool 1 2
@attribute ?sced 1 2 3
```

B.1.4 NASA Orbital Space Plane 2 (More Limited Scope)

```
@project
@attribute prec 4
@attribute ?pmat 4 5
@attribute docu 3
@attribute ?ltex 2 3 4 5
@attribute ?sced 2 3 4
@attribute flex 3
```

```
@attribute resl 4
@attribute team 3
@attribute time 3
@attribute stor 3
@attribute data 4
@attribute pvol 3
@attribute ruse 4
@attribute rely 5
@attribute acap 4
@attribute pcap 3
@attribute pcon 3
@attribute apex 4
@attribute plex 4
@attribute tool 5
@attribute cplx 4
@attribute site 6
```

B.2 NASA93 Historical Data for Defects, Effort, and Months

```
@relation BFC2–NASA93

@attribute prec 4
@attribute flex 4
@attribute resl 4
@attribute team 5
@attribute pmat 2 3 4
@attribute rely 2 3 4 5
@attribute data 2 3 4 5
@attribute cplx 2 3 4 5 6
@attribute ruse 3
@attribute docu 3
@attribute time 3 4 5 6
@attribute stor 3 4 5 6
@attribute pvol 2 3 4
@attribute acap 3 4 5
@attribute pcap 3 4 5
@attribute pcon 3
@attribute apex 2 3 4 5
@attribute plex 1 2 3 4
```

```
@attribute ltex 1 2 3 4
@attribute tool 3 4
@attribute site 3
@attribute sced 2 3
@attribute kloc 0.9 2.2 3 3.5 5.5 6 6.2 6.5 7.25 7.5 7.7 8 8.2 9.7 10 10.4 11.3 11.4 12.8 13 14
    15 15.4 16 16.3 19.3 19.7 20 21 24 24.6 25.9 29.5 31.5 32 32.5 32.6 34 35.5 38 40 41 47.5
    48.5 50 53 60 65 66.6 70 78 79 85 90 98 100 101 111 137 144 150 151 162 165 177.9 190 219 227
     233 240 271 282.1 284.7 302 339 350 352 423 980
@class effort 8.4 10.8 12 18 24 25.2 31.2 36 38 42 48 50 60 62 70 72 82 90 97 98.8 107 114 117.6
    120 150 155 162 170 192 210 215 239 240 252 278 300 324 352.8 360 370 400 409 420 430 432 444
     458 480 571.4 576 599 600 636 648 703 720 750 756 882 973 1181 1200 1248 1350 1368 1645.9
    1772.5 1924.5 2120 2400 2460 4178.2 4560 8211
@class defects 28 69 109 172 188 226 231 240 256 290 302 324 406 420 427 437 456 470 477 566 575
    614 626 636 683 704 765 767 808 810 813 887 920 933 986 1058 1191 1219 1253 1276 1553 1555
    1594 1619 1763 2004 2007 2077 2102 2227 2327 2404 2409 2468 2658 2685 2743 2832 2950 2984
    3340 3343 4210 4256 4342 4511 4815 4840 4868 4907 5092 5434 5848 6129 6136 6266 6293 7553
    7867 7998 8477 8518 8543 8547 8848 9308 9820 10313 11761 17597 18447 50961
@class months 4.9 6.6 7.8 9.1 9.9 10.1 10.4 11.0 11.2 12.0 12.4 12.5 12.8 13.6 13.9 14.4 14.5
    14.8 15.0 15.1 15.2 15.3 15.4 15.5 15.6 16.0 16.2 16.4 17.6 18.6 18.7 18.9 19.2 19.3 20.2
    20.8 21.0 21.3 21.4 21.5 22.3 23.0 23.2 23.5 24.4 24.9 25.0 25.2 25.4 26.2 26.7 26.9 27.5
    28.0 28.8 29.6 30.1 30.3 30.5 31.5 32.2 32.4 32.5 33.6 33.8 34.2 34.5 35.4 35.7 36.2 37.1
    37.3 38.1 38.4 41.9 42.8 42.9 43.4 45.9 47.3 53.1 96.4

@data
4 4 4 5 4 4 2 4 3 3 3 3 2 3 3 3 3 3 4 3 3 2 25.9 117.6 808 15.3
4 4 4 5 4 4 2 4 3 3 3 3 2 3 3 3 3 3 4 3 3 2 24.6 117.6 767 15.0
4 4 4 5 4 4 2 4 3 3 3 3 2 3 3 3 3 3 4 3 3 2 7.7 31.2 240 10.1
4 4 4 5 4 4 2 4 3 3 3 3 2 3 3 3 3 3 4 3 3 2 8.2 36 256 10.4
4 4 4 5 4 4 2 4 3 3 3 3 2 3 3 3 3 3 4 3 3 2 9.7 25.2 302 11.0
4 4 4 5 4 4 2 4 3 3 3 3 2 3 3 3 3 3 4 3 3 2 2.2 8.4 69 6.6
4 4 4 5 4 4 2 4 3 3 3 3 2 3 3 3 3 3 4 3 3 2 3.5 10.8 109 7.8
4 4 4 5 4 4 2 4 3 3 3 3 2 3 3 3 3 3 4 3 3 2 66.6 352.8 2077 21.0
4 4 4 5 4 4 2 4 3 3 6 6 2 4 4 3 4 3 4 4 3 3 7.5 72 226 13.6
4 4 4 5 3 3 2 4 3 3 3 3 2 4 5 3 5 3 4 3 3 3 20 72 566 14.4
4 4 4 5 3 3 2 4 3 3 3 3 2 4 4 3 5 3 4 3 3 3 6 24 188 9.9
4 4 4 5 3 3 2 4 3 3 3 3 2 4 5 3 5 3 4 3 3 3 100 360 2832 25.2
4 4 4 5 3 3 2 4 3 3 3 3 2 4 3 3 5 3 2 3 3 3 11.3 36 456 12.8
4 4 4 5 3 3 2 4 3 3 3 3 4 4 4 3 4 2 1 3 3 3 100 215 5434 30.1
4 4 4 5 3 3 2 4 3 3 3 3 2 4 4 3 5 3 4 3 3 3 20 48 626 15.1
```

4 4 4 5 3 3 2 4 3 3 3 3 2 4 3 3 3 3 1 3 3 3 100 360 4342 28.0

4 4 4 5 3 3 2 4 3 3 3 6 2 4 5 3 5 3 4 3 3 3 150 324 4868 32.5

4 4 4 5 3 3 2 4 3 3 3 3 2 4 4 3 4 3 4 3 3 3 31.5 60 986 17.6

4 4 4 5 3 3 2 4 3 3 3 3 2 4 4 3 5 3 4 3 3 3 15 48 470 13.6

4 4 4 5 3 3 2 4 3 3 3 6 2 4 3 3 4 3 4 3 3 3 32.5 60 1276 20.8

4 4 4 5 4 4 2 4 3 3 3 3 2 3 3 3 3 3 4 3 3 2 19.7 60 614 13.9

4 4 4 5 4 4 2 4 3 3 3 3 2 3 3 3 3 3 4 3 3 2 66.6 300 2077 21.0

4 4 4 5 4 4 2 4 3 3 3 3 2 3 3 3 3 3 4 3 3 2 29.5 120 920 16.0

4 4 4 5 3 4 3 3 3 3 4 3 3 3 4 3 4 3 3 3 3 3 15 90 575 15.2

4 4 4 5 3 4 3 4 3 3 3 3 3 3 4 3 4 3 3 3 3 3 38 210 1553 21.3

4 4 4 5 3 3 3 3 3 3 3 3 3 3 4 3 4 3 3 3 3 3 10 48 427 12.4

4 4 4 5 4 3 5 4 3 3 5 5 2 5 3 3 4 2 4 3 3 2 15.4 70 765 14.5

4 4 4 5 4 3 5 4 3 3 5 5 2 5 3 3 4 2 4 3 3 2 48.5 239 2409 21.4

4 4 4 5 4 3 5 4 3 3 5 5 2 5 3 3 4 2 4 3 3 2 16.3 82 810 14.8

4 4 4 5 4 3 5 4 3 3 5 5 2 5 3 3 4 2 4 3 3 2 12.8 62 636 13.6

4 4 4 5 4 3 5 4 3 3 5 5 2 5 3 3 4 2 4 3 3 2 32.6 170 1619 18.7

4 4 4 5 4 3 5 4 3 3 5 5 2 5 3 3 4 2 4 3 3 2 35.5 192 1763 19.3

4 4 4 5 4 4 2 4 3 3 3 3 2 3 3 3 3 3 4 3 3 2 5.5 18 172 9.1

4 4 4 5 4 4 2 4 3 3 3 3 2 3 3 3 3 3 4 3 3 2 10.4 50 324 11.2

4 4 4 5 4 4 2 4 3 3 3 3 2 3 3 3 3 3 4 3 3 2 14 60 437 12.4

4 4 4 5 3 4 3 4 3 3 3 3 3 3 3 3 3 3 3 3 3 3 6.5 42 290 12.0

4 4 4 5 3 3 3 4 3 3 3 3 3 3 3 3 3 3 3 3 3 3 13 60 683 14.8

4 4 4 5 4 3 3 4 3 3 3 3 3 3 4 3 3 3 4 4 3 3 90 444 3343 26.7

4 4 4 5 3 3 3 4 3 3 3 3 3 3 3 3 3 3 3 3 3 3 8 42 420 12.5

4 4 4 5 3 3 3 4 3 3 4 3 3 3 3 3 3 3 3 3 3 3 16 114 887 16.4

4 4 4 5 4 3 4 4 3 3 5 4 2 4 4 3 3 2 4 3 3 2 177.9 1248 7998 31.5

4 4 4 5 4 4 2 4 3 3 3 3 2 3 4 3 3 3 3 3 3 3 302 2400 8543 38.4

4 4 4 5 4 3 4 2 3 3 3 3 4 4 3 3 4 3 3 4 3 3 282.1 1368 9820 37.3

4 4 4 5 4 4 4 2 3 3 3 3 3 4 3 3 4 3 3 3 3 3 284.7 973 8518 38.1

4 4 4 5 3 4 4 3 3 3 3 3 2 3 4 3 4 3 4 3 3 3 79 400 2327 26.9

4 4 4 5 2 2 3 3 3 3 3 3 2 4 5 3 4 3 4 3 3 3 423 2400 18447 41.9

4 4 4 5 4 3 3 3 3 3 3 3 2 4 5 3 5 2 4 3 3 3 190 420 5092 30.3

4 4 4 5 4 3 3 4 3 3 3 4 3 4 3 3 4 3 4 3 3 3 47.5 252 2007 22.3

4 4 4 5 2 5 3 6 3 3 4 4 2 3 3 3 4 3 3 4 3 3 21 107 1058 21.3

4 4 4 5 2 3 4 4 3 3 5 3 3 4 4 3 4 3 4 3 3 3 78 571.4 4815 30.5

4 4 4 5 2 3 4 4 3 3 5 3 3 4 4 3 4 3 4 3 3 3 11.4 98.8 704 15.5

4 4 4 5 2 3 4 4 3 3 5 3 3 4 4 3 4 3 4 3 3 3 19.3 155 1191 18.6

4 4 4 5 2 4 3 5 3 3 4 4 2 4 3 3 3 4 4 3 3 3 101 750 4840 32.4

4 4 4 5 2 4 3 4 3 3 4 4 2 3 3 3 4 3 3 3 3 3 219 2120 11761 42.8

4 4 4 5 2 4 3 4 3 3 4 4 2 3 3 3 4 3 3 3 3 3 50 370 2685 25.4

4 4 4 5 4 5 4 4 3 3 5 5 3 5 5 3 5 3 4 4 3 2 227 1181 6293 33.8

4 4 4 5 4 3 4 5 3 3 3 3 2 4 5 3 3 2 3 3 3 2 70 278 2950 20.2

4 4 4 5 4 4 2 4 3 3 3 3 2 3 3 3 3 3 4 3 3 2 0.9 8.4 28 4.9

4 4 4 5 2 5 2 6 3 3 6 5 2 4 4 3 5 1 4 3 3 3 980 4560 50961 96.4

4 4 4 5 3 3 2 4 3 3 3 3 2 5 5 3 3 4 4 3 3 3 350 720 8547 35.7

4 4 4 5 4 4 3 6 3 3 4 4 2 4 3 3 3 4 4 4 3 3 70 458 2404 27.5

4 4 4 5 4 4 3 6 3 3 4 4 2 4 3 3 3 4 4 4 3 3 271 2460 9308 43.4

4 4 4 5 3 3 3 3 3 3 3 3 2 4 4 3 4 3 4 3 3 3 90 162 2743 25.0

4 4 4 5 3 3 3 3 3 3 3 3 2 4 4 3 4 3 4 3 3 3 40 150 1219 18.9

4 4 4 5 3 4 3 4 3 3 4 3 2 4 4 3 4 3 4 3 3 3 137 636 4210 32.2

4 4 4 5 3 4 3 4 3 3 4 3 4 4 4 3 4 3 4 3 3 3 150 882 5848 36.2

4 4 4 5 3 5 3 4 3 3 4 3 2 4 4 3 4 3 4 3 3 3 339 444 8477 45.9

4 4 4 5 3 2 4 2 3 3 3 3 4 4 4 3 4 3 4 3 3 3 240 192 10313 37.1

4 4 4 5 2 4 3 4 3 3 3 5 2 4 4 3 4 4 4 3 3 2 144 576 6129 28.8

4 4 4 5 2 3 2 3 3 3 3 5 2 4 4 3 4 4 4 3 3 2 151 432 6136 26.2

4 4 4 5 2 3 2 4 3 3 3 5 2 4 4 3 4 4 4 3 3 2 34 72 1555 16.2

4 4 4 5 2 3 3 4 3 3 3 5 2 4 4 3 4 4 4 3 3 2 98 300 4907 24.4

4 4 4 5 2 3 3 4 3 3 3 5 2 4 4 3 4 4 4 3 3 2 85 300 4256 23.2

4 4 4 5 2 3 2 3 3 3 3 5 2 4 4 3 4 4 4 3 3 2 20 240 813 12.8

4 4 4 5 2 3 2 3 3 3 3 5 2 4 4 3 4 4 4 3 3 2 111 600 4511 23.5

4 4 4 5 2 4 5 4 3 3 3 5 2 4 4 3 4 4 4 3 3 2 162 756 7553 32.4

4 4 4 5 2 4 4 5 3 3 3 5 2 4 4 3 4 4 4 3 3 2 352 1200 17597 42.9

4 4 4 5 2 4 3 5 3 3 3 5 2 4 4 3 4 4 4 3 3 2 165 97 7867 31.5

4 4 4 5 4 4 3 5 3 3 4 4 2 4 3 3 3 4 4 3 3 3 60 409 2004 24.9

4 4 4 5 4 4 3 5 3 3 4 4 2 4 3 3 3 4 4 3 3 3 100 703 3340 29.6

4 4 4 5 3 4 5 5 3 3 6 6 4 3 3 3 3 2 2 3 3 3 32 1350 2984 33.6

4 4 4 5 4 4 4 4 3 3 5 6 4 4 4 3 4 4 4 3 3 3 53 480 2227 28.8

4 4 4 5 4 4 2 5 3 3 5 6 2 5 5 3 5 1 1 4 3 3 41 599 1594 23.0

4 4 4 5 4 4 2 5 3 3 5 6 2 5 5 3 5 1 1 4 3 3 24 430 933 19.2

4 4 4 5 4 5 4 5 3 3 6 6 3 4 4 3 4 4 4 3 3 3 165 4178.2 6266 47.3

4 4 4 5 4 5 4 5 3 3 6 6 3 4 4 3 4 4 4 3 3 3 65 1772.5 2468 34.5

4 4 4 5 4 5 4 5 3 3 6 6 3 4 4 3 4 4 4 3 3 3 70 1645.9 2658 35.4

4 4 4 5 4 5 4 6 3 3 6 6 3 4 4 3 4 4 4 3 3 3 50 1924.5 2102 34.2

4 4 4 5 2 5 2 5 3 3 5 6 2 4 3 3 2 1 2 4 3 3 7.25 648 406 15.6

4 4 4 5 4 5 4 5 3 3 6 6 3 4 4 3 4 4 4 3 3 3 233 8211 8848 53.1

4 4 4 5 3 4 3 5 3 3 5 5 4 3 3 3 3 2 2 3 3 3 16.3 480 1253 21.5

4 4 4 5 3 4 3 5 3 3 5 5 4 3 3 3 3 2 2 3 3 3 6.2 12 477 15.4

4 4 4 5 3 4 3 5 3 3 5 5 4 3 3 3 3 2 2 3 3 3 3 3 38 231 12.0

Bibliography

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

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

[3] J.H. Andrews, F.C.H. 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.

[4] Mohammad Azzeh, Daniel Neagu, and Peter 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.

[5] Dan Baker. A hybrid approach to expert and model-based effort estimation. Master's thesis, Lane Department of Computer Science and Electrical Engineering, West Virginia University, 2007. Available from https://eidr.wvu.edu/etd/documentdata.eTD?documentid=5443.

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

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

[8] 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`.

[9] Barry Boehm, Barry Boehm, and Hoh In. Software cost option strategy tool (s-cost). In *In Conflict Analysis and Negotiation Aids for Cost-Quality Requirements Annual International Computer Software and Applications Conference), IEEE Comp*, pages 15–20. Society Press, 1996.

[10] Barry Boehm, Ellis Horowitz, Ray Madachy, Donald Reifer, Bradford K. Clark, Bert Steece, A. Winsor Brown, Sunita Chulani, and Chris Abts. *Software Cost Estimation with Cocomo II*. Prentice Hall, 2000.

[11] Adam Brady and Tim Menzies. Case-based reasoning vs parametric models for software quality optimization. In *PROMISE'10*, 2010. Available from `http://menzies.us/pdf/10cbr.pdf`.

[12] Adam Brady, Tim Menzies, Oussama El-Rawas, Ekrem Kocaguneli, and Jacky Keung. Case-based reasoning for reducing software development effort. *Journal of Software Engineering and Applications*, 3, 2010. Available from `http://menzies.us/pdf/10w0.pdf`.

[13] F. P. Brooks. *The Mythical Man-Month, Anniversary edition*. Addison-Wesley, 1975.

[14] S. Chulani, B. Boehm, and B. Steece. Bayesian analysis of empirical software engineering cost models. *IEEE Transaction on Software Engineerining*, 25(4), July/August 1999.

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

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

[17] Oussama El-Rawas and Tim Menzies. A second look at faster, better, cheaper. *Innovations in Systems and Software Engineering*, 2011.

[18] N. E. Fenton and M. Neil. A critique of software defect prediction models. *IEEE Transactions on Software Engineering*, 25(5):675–689, 1999. Available from `http://citeseer.nj.nec.com/fenton99critique.html`.

[19] Norman E. Fenton, Martin Neil, and Jose Galan Caballero. Using ranked nodes to model qualitative judgments in bayesian networks. *IEEE Trans. on Knowl. and Data Eng.*, 19(10):1420–1432, 2007.

[20] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[21] 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`.

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

[23] Mark Harman and Joachim 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.

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

[25] 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`.

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

[27] C.F. Kemerer. An empirical validation of software cost estimation models. *Communications of the ACM*, 30(5):416–429, May 1987.

[28] Taghi M. Khoshgoftaar and Naeem Seliya. Fault prediction modeling for software quality estimation: Comparing commonly used techniques. *Empirical Software Engineering*, 8(3):255–283, 2003.

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

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

[31] Ekrem Kocaguneli, Gregory Gay, Tim Menzies, Ye Yang, and Jacky W. Keung. When to use data from other projects for effort estimation. In *IEEE ASE'10*, 2010. Available from `http://menzies.us/pdf/10other.pdf`.

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

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

[34] David Leake and David Mcsherry. Intro. to the special issue on explanation in case-based reasoning. *AI Review*, 24:103–108, 2005.

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

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

[37] U. Lipowezky. Selection of the optimal prototype subset for 1-NN classification. *Pattern Recognition Letters*, 19:907918, 1998.

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

[39] Michael R. Lowry. Towards predictive models of technology impact on software design productivity. 2010.

[40] Emilia Mendes, Ian D. Watson, Chris Triggs, Nile Mosley, and Steve Counsell. A comparative study of cost estimation models for web hypermedia applications. *Empirical Software Engineering*, 8(2):163–196, 2003.

[41] 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`.

[42] T. Menzies, O. Elrawas, D. Baker, J. Hihn, and K. Lum. On the value of stochastic abduction (if you fix everything, you lose fixes for everything else). In *International Workshop on Living with Uncertainty (an ASE'07 co-located event)*, 2007. Available from `http://menzies.us/pdf/07fix.pdf`.

[43] 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`.

[44] 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`.

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

[46] 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`.

[47] 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`.

[48] Tim Menzies, Zhihao Chen, Dan Port, and Jairus Hihn. Simple software cost estimation: Safe or unsafe? In *Proceedings, PROMISE workshop, ICSE 2005*, 2005. Available from `http://menzies.us/pdf/05safewhen.pdf`.

[49] Tim Menzies, David Raffo, Siri on Setamanit, Ying Hu, and Sina Tootoonian. Model-based tests of truisms. In *Proceedings of IEEE ASE 2002*, 2002. Available from `http://menzies.us/pdf/02truisms.pdf`.

[50] Tim Menzies and Forrest Shull. The quest for convincing evidence. In A. Oram and G.Wilson, editors, *Making Software: What Really Works and We We Believe it*. O'Reilly Books, 2010.

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

[52] T.J. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener. Defect prediction from static code features: Current results, limitations, new approaches. *Automated Software Engineering*, (4), December 2010. Available from `http://menzies.us/pdf/10which.pdf`.

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

[54] Y. Miyazaki, M. Terakado, K. Ozaki, and H. Nozaki. Robust regression for developing software estimation models. *J. Syst. Softw.*, 27(1):3–16, 1994.

[55] Martin Možina, Janez Demšar, Michael Kattan, and Blaž Zupan. Nomograms for visualization of naive bayesian classifier. In *PKDD '04: Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 337–348, New York, NY, USA, 2004. Springer-Verlag New York, Inc.

[56] An Ngo-The and G. Ruhe. Optimized resource allocation for software release planning. *Software Engineering, IEEE Transactions on*, 35(1):109–123, Jan.-Feb. 2009.

[57] 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`.

[58] Thomas J. Ostrand, Elaine J. Weyuker, and Robert M. Bell. Where the bugs are. In *ISSTA '04: Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*, pages 86–96, New York, NY, USA, 2004. ACM.

[59] Parag C. Pendharkar, Girish H. Subramanian, and James A. Rodger. A probabilistic model for predicting software development effort. *IEEE Trans. Softw. Eng.*, 31(7):615–624, 2005.

[60] Dietmar Pfahl and Ioana Rus. Special issue on prosim 2004. *Software Process: Improvement and Practice*, 10(3):251–253, July/September.

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

[62] Stuart J. Russell, Peter Norvig, John F. Candy, Jitendra M. Malik, and Douglas D. Edwards. *Artificial intelligence: a modern approach*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2003.

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

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

[65] Thomas Schulz, Lukasz Radlinski, Thomas Gorges, and Wolfgang Rosenstiel. Defect cost flow model: a bayesian network for predicting defect correction effort. In *PROMISE '10*, pages 16:1–16:11, 2010.

[66] Bart Selman, Henry A. Kautz, and Bram Cohen. Local search strategies for satisfiability testing. In Michael Trick and David Stifler Johnson, editors, *Proceedings of the Second DIMACS Challange on Cliques, Coloring, and Satisfiability*, Providence RI, 1993.

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

[68] 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.

[69] M. J. Shepperd. Case-based reasoning and software engineering. Technical Report TR02-08, Bournemouth University, UK, 2002.

[70] H. Gall E. Giger T. Zimmermann, N. Nagappan and B. Murphy. Cross-project defect prediction. In *ESEC/FSE'09*, August 2009.

[71] A. Tosun, A. Bener, and R. Kale. Ai-based software defect predictors: Applications and benefits in a case study. In *Twenty-Second IAAI Conference on Artificial Intelligence*, 2010.

[72] Fiona Walkerden and Ross Jeffery. An empirical study of analogy-based software effort estimation. *Empirical Softw. Engg.*, 4(2):135–158, 1999.

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

[74] Ian H. Witten and Eibe Frank. *Data mining. 2nd edition*. Morgan Kaufmann, Los Altos, US, 2005.

[75] David H. Wolpert and R. Waters. The relationship between pac, the statistical physics framework, the bayesian framework, and the vc framework. In *Proc. SFI/CNL Workshop Formal Approaches to Supervised Learning*, pages 117–214. Addison-Wesley, 1994.

[76] Y. Zhang, M. Harman, and S.A. Mansouri. The multi-objective next release problem. In *In ACM Genetic and Evolutionary Computation Conference (GECCO 2007)*, page 11, 2007.