Learning Project Management Decisions: A Case Study with Case-Based Reasoning Versus Data Farming

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

Abstract—
BACKGROUND: Given information on just a few prior projects, how to learn best and fewest changes for current projects?
AIM: To conduct a case study comparing two ways to recommend project changes. (1) *Data farmers* use Monte Carlo sampling to survey and summarize the space of possible outcomes. (2) *Case-Based Reasoners* (CBR) explore the neighborhood around test instances.
METHOD: We applied a state-of-the data farmer (SEESAW) and a CBR tool ($\mathcal{W}2$) to software project data.
RESULTS: CBR with $\mathcal{W}2$ was more effective than SEESAW's data farming for learning best and recommend project changes, effectively reduces runtime, effort and defects. Further, CBR with $\mathcal{W}2$ was comparably easier to build, maintain, and apply in novel domains especially on noisy data sets.
CONCLUSION: Use CBR tools like $\mathcal{W}2$ when data is scarce or noisy or when project data can not be expressed in the required form of a data farmer.
FUTURE WORK: This study applied our own CBR tool to several small data sets. Future work could apply other CBR tools and data farmers to other data (perhaps to explore other goals such as, say, minimizing maintenance effort).

Index Terms—Search-based software engineering, Case-based reasoning, data farming, COCOMO

1 INTRODUCTION

In the age of Big Data and cloud computing, it is tempting to tackle problems using:

- A *data-intensive* Google-style collection of gigabytes of data; or, when that data is missing ...
- A *CPU-intensive* data farming analysis; i.e. Monte Carlo sampling [1] to survey and summarize the space of possible outcomes (for details on data farming, see §2).

For example, consider a software project manager trying to

- Reduce project defects in the delivered software;
- Reduce project development effort

How can a manager find and assess different ways to address these goals? It may not be possible to answer this question via *data-intensive methods*. Such data is inherently hard to access. For example, as discussed in §2.2, we may never have access to large amounts of software process data.

As to the *cpu-intensive approaches*, we have been exploring data farming for a decade [2] and, more recently, cloud computing [3], [4]. Experience shows that cpu-intensive methods may not be appropriate for all kinds of problems and may introduce spurious correlation under certain situations.

In this paper, we document that experience. The experiments of this paper benchmark our SEESAW data farming tool proposed in [5]–[12] against a lightweight case-based reasoner (CBR) called $\mathcal{W}2$ [13], [14].

If we over-analyze scarce data (such as the software process data of §2.2) then we run into the risk of drawing conclusions based on insufficient background supporting data. Such conclusions will perform poorly on future examples. Our experience shows that the SEESAW data farming tool suffers from many "optimization failures" where if some test set is treated with SEESAW's recommendations, then some aspect of that treated data actually gets worse. On contrary $\mathcal{W}2$ has far fewer optimization failures.

Based on those experiments, this paper will conclude that when reasoning about changes to software projects:

1) Use data farming in data rich-domains (e.g. when reasoning about thousands of inspection reports on millions of lines of code [15]) and when the data is not noisy and when the software project data can be expressed in the same form as the model inputs;
2) Otherwise, use CBR methods such as our $\mathcal{W}2$ tool.

- *Tim Menzies (corresponding author) Adam Brady, Phillip Green and Oussama El-Rawas are with the Lane Department of Computer Science and Electrical Engineering, West Virginia University. E-mail: tim@menzies.us, adam.m.brady@gmail.com, deathcheese@yahoo.com, orawas@gmail.com*
- *Jacky Keung is with the Department of Computer Science, The City University of Hong Kong, Hong Kong SAR E-mail: jacky.keung@cityu.edu.hk.*
- *Steven Williams is with the School of Informatics and Computing Indiana University, Bloomington. E-mail: stevencwilliams@gmail.com.*
- *Jairus Hihn is with Caltech's Jet Propulsion Laboratory. E-mail: jairus.hihn@jpl.nasa.gov.*
- *Barry Boehm is with the University of Southern California. E-mail: boehm@sunset.usc.edu.*

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

Digital Object Indentifier 10.1109/TSE.2013.43

The rest of this paper is structured as follows. §2 discusses data farming, its use in software engineering, and our own data farmer tool: STAR&SEESAW. §3 discusses CBR and $\mathcal{W}2$. These two tools are experimentally compared in §4, which allows us to reflect and assess data farming vs CBR, §5. §6 discusses related work. §7 provides the discussion and future directions to this research.

Before beginning, we emphasis the specific scope of this study. This paper compares one particular data farmer called SEESAW with another particular CBR tool called $\mathcal{W}2$. We will recommend CBR over data farming, under the specific conditions listed at the end of the previous page. It is important to note that this paper does not compare *all* CBR tools to *all* data farming tools. Hence, this is a *case study paper* rather than, say, some formal proof with universal generality.

Nevertheless, it is insightful to compare SEESAW vs $\mathcal{W}2$:

- SEESAW is a complex tool, developed over many years, while $\mathcal{W}2$ was a quick prototype, built very rapidly.
- That is, we are comparing our most complex data farmer against our simplest CBR tool.

The experiments of this paper endorse the simpler CBR tool since it out-performs the more complex data farming tool.

1.1 Relationship to Previous Work

Most of our prior work has been concerned with predictive modeling in software engineering (e.g. [16], [17]). Here, the task is different: we do not *predict* the properties of software projects, but we seek a *plan* to change the project so to improve that prediction. Until very recently, our preferred planning methods was based on either:

- *Data-intensive* machine learning that need at least hundreds of prior examples [18], [19];
- Or *cpu-intensive* data farming [5]–[12].

This paper explores a third approach that is neither data-intensive nor cpu-intensive.

The $\mathcal{W}2$ CBR algorithm introduced in this paper is an extension of two prior instance-based algorithms. $\mathcal{W}0$ [13] was a initial quick proof-of-concept prototype that performed no better than a traditional simulated annealing (SA) algorithm. $\mathcal{W}1$ [14] improved $\mathcal{W}0$'s ranking scheme with a Bayesian method. With this improvement, $\mathcal{W}1$ performed at least as well as a state-of the art model-based method (the SEESAW algorithm discussed below)[1]. $\mathcal{W}2$ improved $\mathcal{W}1$'s method for selecting related examples. With that change, $\mathcal{W}2$ now outperforms state-of-the art model-based methods.

Apart from the new experiments with $\mathcal{W}2$ reported in this paper (see §3.3 and §4), the other material that has not appeared before is the background material of §2.1,§2.2; the detailed description of $\mathcal{W}2$ in §3.1.1, §3.2; and the discussion material of §5, §6.

1. To defend the claim that SEESAW is a state-of-the-art tool, we note that papers discussing SEESAW and its development have appeared in research forums with extensive peer review such as the international conference on SE (ICSE) [8]; the automated SE conference (ASE) [11], [20]; the ICSP software process conference [7]; and peer-reviewed journals [9], [12]. Note that none of those reviewers proposed an obvious better method than SEESAW.

2 Data Farming

2.1 Background

Data Farming is a model-based activity that reflects over a model learned from data:

1) *Plant a seed*: build a model from domain information. All uncertainties or missing parts of the domain information are modeled as a space of possibilities.
2) *Grow the data*: execute model and, when the model moves into the space of possibilities, output is created by selecting at random from the possibilities.
3) *Harvest*: summarize the grown data via data mining.
4) Optionally, use the harvested summaries to improve the model, then go to 1 [21].

Data farming has a long tradition in computer science- its origins dates back to the first experiments with Monte Carlo analysis on the Princeton computers in the early 1950s [22]. Rosenbluth showed that the sampling favors the more probable choices then within a Monte Carlo simulation, any deviations from the non-canonical distributions die away. That is, this kind of Monte Carlo sampling correctly converges on the underlying distribution [1].

Rosenbluth et al. applied Metropolis sampling to many problems like the design of nuclear weaponry. Another application of data farming is in the social sciences. The statistician Nate Silver used this method to successfully predicted the results of the 2008 and 2012 American presidential election. His data farmers repeatedly sampled the distributions seen in polling data to compute the probability of politicians winning in different seats [23]. Note that, with Rosenbluth and Silver, this work combined extensive simulation with extensive data collection (i.e. Silver collected data from many voters while Rosenbluth and his colleagues tuned their models via data collected from extensive experimentation).

Data farming has been used extensively by the U.S. Military [24] and, as shown below, in software engineering. Data farming builds a "landscape" of output that can be analyzed for trends, anomalies, and insights in multiple parameter dimensions. Figure 1 shows the landscape around some NASA project data. This landscape is particularly useful for extrapolating from the older projects to new projects.

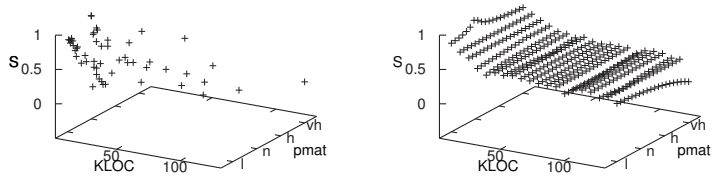

Fig. 1: In data farming, past project data (shown on left) is used to develop a software process model. This model is then executed to generate a landscape of possibilities (on right). In this figure, pmat is process maturity; KLOC is lines of code; "*S*"= a score function that rewards projects with low defects and fast development times (so *larger* "*S*" scores are *better*). For an example of this function, see Equation 1 shown later in this paper.

The drawback with data farming is that models can make predictions outside of the space of data seen in past projects. For example, in Figure 1, most of the data is clustered around *low KLOC* and *low* to *nominal* process maturity. Outside of that region, project data is sparser and the conclusion from a data farmer can be less reliable (such reliability is less manifest in the work of (say) Rosenbluth or Silver since the simulations are used in conjunction with extensive data collection).

The problem of drawing conclusions from sparse data is particularly acute with software process data. As discussed in the next section, we have tried unsuccessfully for many years to access large databases of software process data. Consequently, all the data sets used in this study are relatively small. Hence, we might expect poor results from data farming:

- *Poor improvement:* When we compare the (say) defect and effort distributions *before* and *after* we apply those recommendations, the median reduction in defect and effort will not be large;
- *Poor control in the treated data:* When we study the (say) defect and effort distributions in the data treated with the recommendation, then the spread of those distributions will be very large.

The aim of this paper is to check if CBR or data farming suffers most from *poor improvement* or *poor control*.

2.2 Software Process Data

Once reason to explore data farming is that it can be used to draw conclusions from small databases of software process data. Given some information about past projects, it is possible to interpolate and extrapolate from the past into the present.

Fenton [25] divides software project data into:

- *Process* measures about the way software is constructed;
- The *resources* used to generate the project;
- The details of the constructed *product*.

It is easier to collect product measurements than process measurements. For example, any open source repository offers many details of its products. However, process and resource information are harder to find. Consider one project running for one year writing hundreds to thousands of classes:

- Each one of those classes may have an extensive inspection log and entries in the issue tracking system. Hence, that project may generate hundreds to thousands of product records.
- On the other hand, that year of work may generate only a single record in a process database.

Figure 2 shows the small size of the data sets used in this paper. Note the small size of those data sets. Small training sets are very common in software process databases. For example, when compared to data sets seen in effort estimation literature, the data of Figure 2 are not unusually small[2].

We do not expect our process data sets to grow larger in the foreseeable future. After 26 years of trying, one of us (Boehm) has collected less than 200 sample projects for the COCOMO database. In other work, even after 2 years of effort, we could add just seven records to a NASA-wide software cost metrics repository [20]. This is due to the business sensitivity associated with the data as well as differences in how the metrics are collected and archived.

2. Five recent effort estimation publications [17], [26]–[29] use data sets with 13,15,31,33,52 rows (respectively).

Data set	Cols	Rows	Notes
Kemerer	7	15	Large business applications
Telecom	3	18	U.K. telecom enhancements
Finnish	8	38	Finnish IS projects
Miyazaki	8	48	Japanese COBOL projects
COC81dem	26	63	NASA projects
NASA93dem	26	93	NASA projects
Median	8	43	

Fig. 2: The data sets used in this study (available on-line at http://promisedata.googlecode.com).

Many researchers have explored the analysis of large sets of product data: e.g. see the proceedings of the following conferences: Mining Software Repositories or PROMISE. However, the analysis of small data sets is less represented in that literature. The rest of this paper discusses methods for making decisions about the small data sets of Figure 2.

2.3 Applications of Data Farming in SE

When historical data is scarce, it is possible to hypothesis a model, then extrapolate examples from that model. Such *model-based data farming* runs a simulation many times across a large parameter and value space. As shown by the following examples, this approach is often applied in SE.

For example, consider the 2CEE effort estimation model [28]. In 2CEE, any uncertainty about project attributes is represented as a probability distribution. The effort estimates shown in Figure 3 result from passing thousands of samples from those input distributions to an effort estimation model. The 2CEE convention is to report to the users the 50th to 70th percentile range of the generated effort estimates. In Figure 3, that range is 853 to 1141 months.

In summary, 2CEE is a Monte Carlo simulator for estimations models, where the inputs are drawn from probability distributions. This kind of analysis is widely-used in the the effort estimation community; e.g. [30]. In fact, on of the most famous examples of data farming in software engineering are

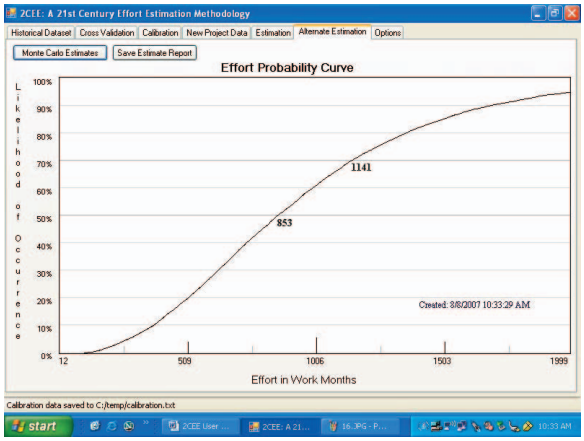

Fig. 3: Simulation output from the 2CEE effort estimation tool [28].

the two IEEE TSE *simulation studies* papers of Myrtveit, Stensrud, Shepperd & Kadoda [31], [32]. Those papers studied the comparative rankings of effort estimation models under different experimental conditions. A challenge faced by those papers was a lack of real-world software project data. Hence, they applied data farming. Artificial data sets were generated by learning distributions for the available data, then sampling from those distributions to auto-generating large data sets.

Pearce [33] used data farming to build monitoring software for satellites from qualitative models of requirements, written in Prolog. Prolog's backtracking mechanism was used to generate large samples of data from the requirements model. This output was summarized by a data miner into a set of rules, which were compared to another rule set, built manually by another team:

- The manual rules took twice as long to build and had more syntactic errors (e.g., type errors and dead-end rules).
- In an analysis of abnormal satellite behaviors generated by a simulator for that satellite, the manual rules predicted far fewer abnormal conditions than the farmed rules.

Another modeling framework used for data farming are goal graphs [34] and Mylopoulos' soft-goal graphs [35]. Soft goals are optional nonfunctional requirements. The task of a soft goal analysis is to satisfy as many of these goals as possible. Chang et al. explored Prolog models of soft-goals [36]. Like Pearce, Chang et al. backtracked through all possible sets of satisfied soft-goal graphs, then asked a data miner to learn the decisions that selected for maximal soft-goal coverage.

Goal graphs are a tree of alternative sets of goals, tagged with the data structures required to achieve each sub-goal. Once the user selects their particular goals, the associated data structures then form the high-level architecture for a system. Heaven & Leiter [37] applied data farming to a quantitative goal graph model of the requirements of the London Ambulance Services [37] (the upper layers of that requirements model were a standard van Lamsweerde goal graph [34] while the leafs drew their values from statistical distributions). During the multiple simulations conducted by a genetic algorithm (GA), choices were evaluated by selecting at random from the leaf distributions.

GAs have been used in other data farming applications. Rodriguez et al. [38] studied systems dynamics models of software projects. They combined a systems model with the NSGA-II genetic algorithm [39] that reviewed prior inputs to propose inputs for the next simulation. That GA searched for inputs that generated outputs which most decreased time and cost while increasing productivity. That study generated a three-dimensional surface where managers can explore trade-offs between the goals of cost, time and productivity.

Yet another approach to data farming was reported by Jiang et al [40]. That study sought patterns in "runaway" projects; i.e. those whose schedule, cost, or functionality was twice as bad as the original estimates. The challenge with that work was that the research team only had access to a very small database of example runaway projects (10 runaways and 22 other projects expressed in the same format). In their *twice-learning* scheme, a random forest of decision trees was generated from the original data. New "virtual examples" were generated by, many times, picking some tree branch and generating an instance consistent with the constraints on that branch. These virtual examples were then summarized by a second decision tree learner.

In summary, data farming has been applied numerous times in software engineering. While not a widely used technique, it is used commonly enough to deserve the serious consideration of the research community.

2.4 Data Farming with STAR & SEESAW

To assess the merits of data farmer vs something else, we need to carefully select the case study for that comparison. This evaluation will be based on our own STAR/SEESAW data farmer (described in this section) and the $\mathcal{W}2$ CBR tool (described in the next section). One advantage of this comparison is that both tools were developed independently to handle the same test data (see Figure 2). Hence, at least at the level of data inputs, the two tools are compatible.

To introduce STAR/SEESAW, we return to Figure 3. When business users see that diagram, they often ask "what causes the difference between the low and high effort projects?". In asking that question, they are seeking ways to *change* their projects so as to complete them in less time.

To answer this question, we built the STAR data farmer [20]. STAR's goal was to find the *smallest set* of inputs ranges that *most improves* model outputs. STAR used Bohem's COCOMO/COQUALMO effort/defect estimation tools [41].

STAR used distributions specific to each project to control the distribution of model inputs. For example, Figure 4 shows the known ranges for a NASA flight guidance system called "Orbital Space Plane" (OSP). When generating model inputs, STAR selected across all the ranges of Figure 4. Note that these ranges are divided into *controllable* and *uncontrollable*, where the latter represent attributes that managers cannot change for the OSP project.

STAR's models use the variables of Figure 5. These models generate estimates for effort, defect, and months. Time is the calendar months from start to finish; effort is the total staff hours over that time; so the recommended number of programmers is $\frac{effort}{time}$. STAR tries to maximize:

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

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

STAR executed by growing a *recommendations* on how to change a project. For example, after analyzing Figure 4, STAR's recommendation was:

$$pmat = 4 \ \wedge \ sced = 3 \ \wedge \ cplx = 5 \wedge$$
$$acap = 3 \ \wedge \ ruse = 2 \ \wedge \ stor = 3$$

STAR's search algorithm (simulated annealing) had operators for taking an existing recommendation, then trying some additions. After each addition, Boehm's models would be called to check if a more complex recommendation was any better than a shorter one. Once that check failed, STAR terminated and returned the best recommendation seen to date.

	controllable			uncontrollable	
project	attribute	low	high	attribute	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		
	tool	2	3		
	sced	1	3		
	cplx	5	6		
	KSLOC	75	125		

Fig. 4: OSP: a JPL flight guidance system prototype for the Orbital Space Plane. The numbers {*1, 2, 3, 4, 5, 6*} map to {*very low, low, nominal, high, very high, extra high*}. Attribute names from Figure 5.

id	attributes
1	pcap: Personnel/team capability
2	cplx: Product complexity
3	acap: Analyst capability
4	time: Time constraint
5	rely: Required software reliability
6	site: Multi-site development
7	docu: Doc. match to life cycle
8	pcan: Personnel continuity
9	aexp: Applications experience
10	tool: Use of software tools
11	pvol: Platform volatility
12	stor: Storage constraint
13	pmat: Process maturity
14	ltex: Language & tools experience
15	sced: Required dev. schedule
16	data: Data base size
17	pexp: Platform experience
18	resl: Arch. & risk resolution
19	prec: Precedentedness
20	reuse: Developed for reuse
21	team: Team cohesion
22	flex: Development flexibility

Fig. 5: Variables in Boehm's COCOMO models [41].

Experience with STAR suggested that the algorithm was wasting much time by exploring useless additions [10]. Notice how, in the above recommendation, all the values are some extreme point of the supplied range; e.g. Figure 4 says *pmat* $\in \{1, 2, 3, 4\}$ and STAR recommended maximum process maturity (*pmat* $= 4$). When we reviewed STAR's recommendations, we noticed that while STAR can sometimes recommend non-extreme values, at least half the time, the algorithm prefers the extremes. This observation lead to the development of SEESAW [8], a search algorithm that tended to select those extreme values.

To comparatively assess SEESAW, we built a generalization of STAR that supported multiple search engines including STAR's simulated annealing and SEESAW and (just to be thorough) the other algorithms of Figure 6: ISSAMP, Beam search, A-STAR, and MaxWalkSat. These algorithms were chosen to include traditional AI search methods (e.g. A-STAR) as well as more recent, state of the art, tools (MaxWalkSat).

Green and Menzies et al. [10] ran these six algorithms on four case studies like Figure 4. Each algorithm was run 20 times (guided by the *value* function of Equation 1) over Boehm's effort, months, and defect estimation models. In that evaluation, the best algorithms were SEESAW and BEAM. While the performance of these two was usually equivalent, BEAM ran 10 times slower than SEESAW. Also, occasionally, SEESAW beat BEAM. Hence, we use SEESAW for comparing data farming to CBR.

Simulated annealing [42] (SA) is a Monte Carlo (MC) sampling algorithm that samples controllable model inputs. The classic MC algorithm is Metropolis [43], which creates new states by small mutations to some *current* state. The algorithm is silent on the mutation mechanism. For our experiments, we freeze $\frac{2}{3}$ of the inputs and randomly select ranges for the rest. In SA, if a new state is "better" (as assessed via an "energy function" such as Equation 1), it becomes the new "current" state used for future mutations. Otherwise, a probabilistic criteria is applied to accept, or reject, the new state: (the worse the new state, the less likely that it becomes the new current state). SA uses a "temperature" variable to the acceptance criteria such that, at high temperatures, it is more likely that the algorithm will jump to a new worst current state. This allows the algorithm to jump out of local minima while sampling the space of options. As the temperature cools, such jumps become less likely and the algorithm reverts to a simple hill climber.

ISSAMP is a fast stochastic sampling method that extends a current set of controlled variables using randomly selected ranges. After finding a solution, ISSAMP resets to the start to try other paths (our ISSAMP uses 20 resets). ISSAMP is remarkably effective at scheduling problems, perhaps because it can rapidly explore more of the search space [44]. To avoid exploring low-value regions, our version of ISSAMP 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.

BEAM search is a tree search algorithm 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 the model input, the missing values were selected at random from the legal ranges of those inputs. 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). F was estimated as the percentage of the project descriptors with ranges in the current branch; Also, g was estimated using $1 - Equation\ 1$ (i.e. distance to the utopia of no effort, no development time, and no defects).

MaxWalkSat: Given a random selected set of model inputs, MaxWalkSat tries n modifications to randomly selected attributes [45]. Sometimes (controlled by the α parameter), the algorithm chooses the range that minimizes the value of the current solution. Other times (at probability $1 - \alpha$), a random range is chosen for the attribute. After N retries, the best solution is returned. Our implementation used $n = 50$, $\alpha = 0.5$, and $N = 10$.

SEESAW [10] augments MaxWalkSat with a search heuristic taken from simplex optimization. Like MaxWalkSat, SEESAW selects each attribute at random, selected on each iteration. SEESAW ignores all ranges except the minimum and maximum values for an attribute.

Fig. 6: Some AI algorithms used for data farming.

3 CASE-BASED REASONING WITH $\mathcal{W}2$

Cohen [46] advises that a supposedly sophisticated system should be baselined against a simpler algorithm. Accordingly, after determining that SEESAW was our best data farming algorithm, we set out to build a simpler alternative: the $\mathcal{W}2$ case-based reasoner (CBR).

CBR matches details of a current project to a library of past completed projects. Decisions about the current project are made by reflecting over similar prior completed projects [47], [48]. CBR does not build an intermediary model of (e.g.) the correlation between lines of code and defects. Rather, all reasoning is a partial match between the new examples and the old examples.

CBR is used extensively in software effort estimation [27], [49]–[59]. There are many reasons for this:

- It works even if the domain data is sparse [31].
- Unlike other predictors, it makes no assumptions about data distributions or an underlying model.

With very few exceptions, much of the research into analyzing project effort such as [17], [27], [29], [49], [51]–[55], [57]–[61] offers algorithms to estimate the effort of the *current* project. Those exceptions include Pendharkar & Rodger [62] who note a *scale effect* in development, such that increasing team size can have a serious detrimental effect on the cost of that project. They propose trade-offs between software effort, team size, and development costs. Their analysis requires some skill in the human operators. On the other hand, the $\mathcal{W}2$ case-based reasoner described below fully automates the process of finding the *least* changes to a project in order to *most* improve it (e.g. reduce defects and effort).

3.1 $\mathcal{W}2$: Design Assumptions

The goal of $\mathcal{W}2$ is to find minimal changes to a project that most improve that project.

Like any CBR system, $\mathcal{W}2$ assumes access to historical *cases* described using P project descriptors (e.g. analyst capability, process maturity, etc).

$\mathcal{W}2$ also assumes that (a) each case is described by a set of qualities such as number of defects, development time, total staff effort etc; and that (b) all these qualities are summarized into a single value by some *value* function such as Equation 1.

Further, $\mathcal{W}2$ assumes that a manager can offer us (a) a description of the *context* $\subseteq P$ that interests them and (b) a list of *control*able options which they can change (*control* $\subseteq$ *context*). For example, once we asked a NASA software project manager for a description of the effects of assigning inexperienced people. The manager commented that, at her site, such inexperience implies low applications experience (*aexp*), low to very low platform experience (*plex*), and language and tool experience (*ltex*) that is not high. Next, we asked the manager to describe the range of projects seen at her site (using the COCOMO names of Figure 5). The resulting $context_1$ is shown below:

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

Here, "?" are the *control*labels; for example, this manager is senior enough to adjust factors like schedule pressure (*sced*). Note that there is no requirement for managers to include all project descriptors in their context statement since $\mathcal{W}2$ can handle *context*s that are a subset of the descriptors.

Finally, $\mathcal{W}2$ assumes that we should not reason on *all* the data. Given a distance measure the distance between a *context* and a case, $\mathcal{W}2$ restricts the reasoning to just the cases closest to the *context*s (thus respecting the *context* limitations offered by the user). A simple Euclidean measure will not work for $\mathcal{W}2$ since our *context* attributes are a multi-set (e.g. in the above, $pcap \in \{3,4,5\}$). Also, in the case where the *context* mentions only a few attributes, then most of the Euclidean distances will have to be approximated using some function that handles missing values. Consequently, we use a set overlap function inspired by Aha *et al.* [63]. The distance between case and a *context* is the size of their *overlap*; i.e. the number of attribute values they have in common:

$$overlap(context, case) = |case \cap context| \tag{2}$$

3.1.1 $\mathcal{W}2$: Pseudo code

This section describes $\mathcal{W}2$'s pseudo code, shown in Figure 7. Any italic number in brackets refers to a line number in Figure 7. For example, the algorithm starts by discretizing all numeric values *(15)* into a fixed number of **bins** (such discretization is recommended practice for dealing with small data sets [64]). Also, any letter in **bold font** refers to globals allocated in the `settings` function of Figure 7. $\mathcal{W}2$'s settings were determined using our engineering judgment. In future studies, we will investigate other settings. For the moment, we comment that the current settings let $\mathcal{W}2$ out-perform the model-based data farming methods discussed above.

$\mathcal{W}2$ inputs *(11)* a data set of projects and a *context*. As output, it prints (a) what treatments it can find and (b) the effects of those treatments on a hold-out test set *(19)*. Each treatment is a conjunction of attributes, where each attribute can take one or more values.

All the reasoning is restricted to data that overlaps the *context*. In order to determine the stability of the learned treatment, the algorithm runs multiple times *(14)* using different randomly selected subsets of the data *(16,17)*.

The code divides the cases randomly into *train:test* in the ratio **T**:**t** *(17)*. Following Quinlan [65], we use two-thirds, one-thirds for training and test. Then, Equation 2 is used to find the neighborhood of the *context (22)*. This neighborhood is divided into those cases with the **B** best scores, and the **R** rest scores. *(25)* Appealing to the central limit theorem, we say that the neighborhood of a *context* are the 20 cases "nearest" to that *context*, where "near" is measured by Equation 2.

```
1 function settings() {
2      T = 67      ;; size of training set
3      t = 100 -  T ;; size of test set
4      N = 20      ;; size of neighbors
5      B = 5       ;; #best items in neighbors
6      R =  N -  B ;; #rest items in neighbors
7      S = 3       ;; min support of an acceptable test
8      repeats  = 20
9      bins  = 5
10 }
11 function w2 (context, dataset) {
12   settings() ;; set control variables
13   while ( repeats > 0)
14      repeats--
15      dataset = discretizeNumerics(dataset, bins)
16      train_cases =  T% of dataset, chosen randomly
17      test_cases  = dataset - train_cases
18      treatments  = train(context, train_cases)
19      print test(context, test_cases, treatments) }
20 }
21 function train (context, train_cases) {
22   neighbors = overlaps(context,train_cases)
23   neighbors = sort neighbors by  Equation 1
24   best = top  B cases from neighbors
25   rest = remaining  R cases from neighbors
26   for attr_val ∈ context {
27       b = frequency of attr_val in best /  B
28       r = frequency of attr_val in rest /  R
29       if b > r {
30            scores[attr_val] = b*b / (b+r) }} ;; Equation 3
31   candidates = list of attr_vals ranked on sorted(scores)
32   return prune(candidates, neighbors)
33 }
34 function overlaps (constraing, data) {
35   for case ∈ data { score[case] = overlap(constraing,case)}
36   return top N cases from sorted(scores)
37 }
38 function overlap(constraing,case) { ;;implements Equation 2
39   for attr_val ∈ case {
40       if attr_val ∈ constraint {
41          n++}}
42   return n
43 }
44 function prune (candidates, neighbors) {
45   median[0] = median(all util in neighbors)
46   spread[0] = spread(all util in neighbors)
47   constrained = neighbors
48   treatments =  empty set
49   i = 0
50   while (treatment = pop(candidates))  {
51       for case ∈ constrained {
52           if not (treatment ∈ case) {
53                constrained = constrained - case
54                if size(constrained) <=  S
55                     goto END}}
56       i++ ; at this point, the treatment is acceptble
57       median[i] = median(all util in constrained)
58       spread[i] = spread(all util in constrained)
59       if median[i] > median[i-1]
60          if  spread[i] > spread[i-1]
61               return treatments ; since no improvement
62       treatments[i] = treatment }
63   return treatments
64 }
65 function test (context, test_cases, treatments) {
66    neighbors = overlaps(context,test_cases)
67    selected  = overlaps(treatments, neighbors)
68    m0   = median(all util in neighbors)
69    m1   = median(all util in selected)
70    s0   = spread(all util in neighbors)
71    s1   = spread(all util in selected)
72    return treatments, (m0 - m1)/m0, (s0 - s1)/s0
73 }
74 function median(x){return x's 50th percentile}
75 function spread(x){return x's (75th -25th) percentile}
```

Fig. 7: Pseudo code for $\mathcal{W}2$. "`Util`" refers to the values generated by Equation 1 for a particular case. Code for `discretizeNumerics` is not shown: that function replaces each value x in columns with numeric attributes with $round((x - min)/((max - min)/bins)$, where *max* and *min* are the maximum and minimum values in that column.

All attribute values are ranked according to how often they appear in best than rest. After a review of the analogy-based estimation literature, [26], [29], [49], [50], [66] we noted that no researcher proposed using more than five neighbors for instance-based effort estimation. Hence, we used $B = 5$. $\mathcal{W}2$ ranks attribute values using a Bayesian ranking scheme *(30)*. If "nominal tool use" (denoted as *tool=nom*) occurs 5 times in the best set and 14 times in rest, then:

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

In order to avoid evidence that is infrequent, but relatively more frequent in best than rest, $\mathcal{W}2$ uses a support term. Such support should *increase* as the frequency of a range *increases*, i.e. $ratio(E|best)$ is a valid support measure. Hence, $\mathcal{W}2$'s range ranking formula (as used at line *29*) is:

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

We have found Equation 3 to be a successful ranking heuristic in many applications such the optimization of NASA requirements models [67] and flight guidance systems [68] as well as other applications. [10], [11], [20]

Once all the attribute values are ranked, $\mathcal{W}2$ applies a greedy search to prune away the least effective *(44)* (this is the *over-fitting avoidance* step used in many data mining algorithms such as C4.5 [65] and RIPPER [69]). Using the neighborhood of the *context* in the training set, $\mathcal{W}2$ applies the top i-th ranked attribute values to select progressively smaller subsets of the cases. This process stops when:

1) It runs out of attribute values *(50)*;
2) Or the selected set is too small *(54)*;
3) Or the scores in the selected subsets stop improving *(59)*.

To define stopping rule #2, we used the machine learning literature. Quinlan [65] blocks sub-tree generation for subsets with 3 or less examples; hence, we used $\mathbf{S} = 3$.

To define stopping rule #3, Equation 1 was applied to selected cases. These scores are reported as either the median value *(74)* or the "spread" a.k.a. inter-quartile range *(75)*.

Finally, to test the generated treatment, $\mathcal{W}2$ applies them to the neighborhood of the *context* in the test cases *(65)*. The results of that test are reported as the reduction percentage $100 * (initial - final)/initial$.

3.2 Complexity

$\mathcal{W}2$ is not a slow algorithm. Given simple indexing, the frequency counts requested in `train` (on lines *27,28*) could take linear time. Similarly, the search for constrained subsets in `prune` (*51*) would also take linear time. Much of the processing is linear on the number of attributes a times the number of **bins** per attribute. The only exception to this is the log linear `sort` of the $a * \mathbf{bins}$ attribute values at line *(31)*. Hence, we say that the complexity of $\mathcal{W}2$ is log linear on the number of attribute values; i.e. $O(a * \mathbf{bins} * log(a * \mathbf{bins}))$.

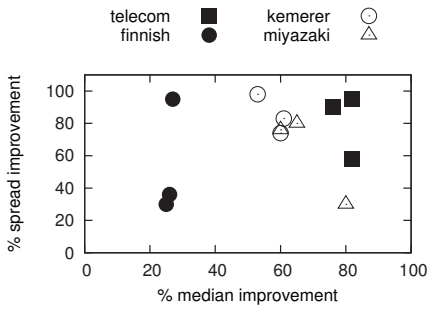

Fig. 8: Effort improvements for four data sets.

3.3 Example

Figure 8 shows the results seen after applying $\mathcal{W}2$ to four of the data sets of Figure 2:

- Enhancements to a U.K. TELECOM product;
- Projects collected by Miyazaki et al [70];
- FINNISH Information Systems projects;
- Large COBOL projects, collected by KEMERER [71].

The attributes in this data are highly varied and includes number of basic logical transactions, query count, and number of distinct business units serviced. For each data set, three *contexts* were generated (so 3*4=12 experiments, in all):

- The first contained the *entire range* of possible project descriptors, representing complete freedom to recommend any change within the space.
- The other two queries were generated by randomly choosing 50% of each attribute values from either the lower, middle, or upper ranges for each project descriptor.

For these experiments, the *values* function was just "reduce effort". The results, shown in Figure 8, are expressed in terms of "improvement" defined as $100 * (\mathit{initial} - \mathit{final})/\mathit{initial}$ (and *large* values are *better* since that indicates a smaller final effort). Here, initial is a measure seen in all the test data while "final" was the measure seen after the rules of $\mathcal{W}2$ were applied. Two measures were used in this study: median and spread of effort; i.e. the 50th and 75th-25th percentile range.

Note that all the points in Figure 8 are positive; i.e. improvements were seen in all cases. While the large median improvements of Figure 8 are encouraging, they may not reflect what is realistically achievable. In these examples, we focused *only* on effort and there are many ways to cut a project's budget with disastrous results (e.g. allocating no effort to testing). Hence, our next experiment will seek to improve *effort*, *defects* and *months*.

4 CBR vs Data Farming

Recall that SEESAW uses Boehm's COCOMO models. That is, any data run through SEESAW must conform to the structure of Figure 5. Accordingly, to test SEESAW and $\mathcal{W}2$ on the same data, we must restrict that test to COCOMO data.

For this comparison, we ran the recommendations generated by SEESAW and $\mathcal{W}2$ on the the NASA93dem and COC81dem data sets of Figure 2. These data sets all have the attributes defined by Boehm [72]; e.g. analyst capability, required software reliability, and use of software tools (for a full list of attributes, see Figure 5). Originally collected in the COCOMO-I format, JPL business experts have translated these data sets from their original COCOMO format to COCOMOII.

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

Fig. 9: Three contexts described by domain experts from JPL [67]. Attribute names from Figure 5. {*1, 2, 3, 4, 5, 6*} denote {*very low, low, nominal, high, very high, extra high*}.

Both SEESAW and $\mathcal{W}2$ guided their search using Equation 1 and the *context*s of Figure 4 and Figure 9:

- OSP: orbital space plane;
- OSP2: a second generation of the OSP software;
- JPL FLIGHT systems;
- JPL GROUND systems.

SEESAW used those *context*s to constrain its exploration of possible recommendations. $\mathcal{W}2$ took those *contexts* then applied the `train` procedure of Figure 7. Recall that, in that procedure, some treatment 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 $\mathcal{W}2$.

To make that comparison, we applied SEESAW and $\mathcal{W}2$'s recommendations (and by "apply", we mean reject any row that contradicts the ranges in the recommendation). Note that this comparison was made on hold-out test data set, not used in any proceeding analysis (for details on the construction of that test set, refer back to Figure 7).

The results of this comparisons are shown in Figure 10, divided into the defect, effort, months changes from GROUND,

Win	Goal	Treatment	Median $a=$ as is	Median $t=$ to be	Spread $A=$ as is	Spread $T=$ to be	Median improv. $\frac{a-t}{a}$	Spread improv. $\frac{A-T}{A}$
NASA93dem 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%
NASA93dem 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%
NASA93dem 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%
NASA93dem 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%
COC81dem 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%
COC81dem 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%
COC81dem 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%
COC81dem OSP2								
*	defects	W	1850	1802	2697	2405	3%	11%
	defects	SEESAW	1473	2269	1769	2061	-54%	-17%
*	effort	W	122	130	431	356	-7%	17%
	effort	SEESAW	98	447	289	288	-356%	0%
	months	SEESAW	19	19	7.9	8.6	-3%	-9%
	months	W	20	21	11	10	-4%	10%

Fig. 10: Changes in median and spread. Asteriks (*) in column one mark a comparison with a better and statistically significantly different median value (Mann- Whitney, 95% confidence). Median= 50th percentile. Spread= 75th-50th percentile range. Gray cells show optimization failures (zero or negative improvement).

FLIGHT, OSP2 & OSP. We show 24 comparisons:

$$\left(\begin{array}{c}\text{NASA93dem}\\ \text{COC81dem}\end{array}\right) * \left(\begin{array}{c}\text{defects}\\ \text{effort}\\ \text{effort}\end{array}\right) * \left(\begin{array}{c}\text{ground}\\ \text{flight}\\ \text{OSP}\\ \text{OSP2}\end{array}\right)$$

$\mathcal{W}2$ produced larger median reductions than 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 and 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 $\mathcal{W}2$).

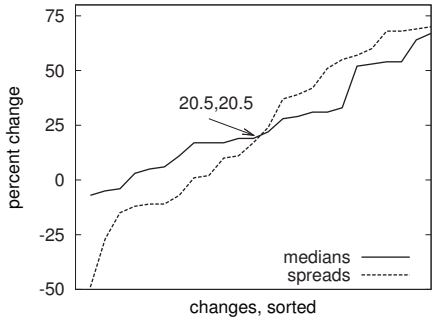

Fig. 11: Range of changes in median and spread generated by applying the recommendations of $\mathcal{W}2$. The median observed changes were (20.5, 20.5)% for (medians, spreads), respectively.

The gray cells in Figure 10 show optimization *failures*; i.e. a zero or negative improvement. $\mathcal{W}2$ had fewest failures: $3/24$ and $7/24$ failures for medians and spreads (respectively), while SEESAW showed $13/24$ and $7/24$ failures for medians and spreads (respectively). One SEESAW failures was particularly dramatic: an increase from 98 effort months to 447 effort months in the OSP2 effort results.

Figure 11 shows the sorts of the median and spread improvements seen from the Figure 10 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 with poor spreads $\mathcal{W}2$ were in the COC81dem data set (an observation we will return to, below).

5 Discussion

This section discusses the issues raised at the end of §2.1, as well as the validity and reliability of our conclusions.

5.1 What methods generate results with *poorer improvements* and *poorer control*?

Recall from Figure 10 that in the majority case ($\frac{16}{24}$), SEESAW resulted in smaller reductions than $\mathcal{W}2$. That is, data farmer resulted in poorer improvements than CBR.

Also recall from Figure 10 that SEESAW had more optimization failures than $\mathcal{W}2$. That is, the recommendations generated by data farmer resulted in poorer control of the treated projects than CBR.

We conjecture that CBR out-performed data farming for the reasons discussed in §2:

- Data farming uses a model to extrapolate a landscape around the data used to develop models.
- Incorrect conclusions result when the landscape extends into poorly sampled regions of the training data.

Our results suggest that, when learning small data sets that only cover a small space of possible projects, it is unwise to

fit one model over all the data (since that model can fail in sparsely sampled local regions). For small data sets, it may be better to extrapolate from just locals region around the test case (in the case of $\mathcal{W}2$, that local region is selected by the *context* query discussed in §3.1).

Also, there is some evidence that CBR may be the preferred approach when data contains noise. Note that most of the gray cells of Figure 10 occur in the COC81dem results. Boehm assumed that this data was to be analyzed by regression so he spent much effort on the COC81dem data (applying his domain expertise to prune or trim outstanding values). Curiously, $\mathcal{W}2$ performed best on the "non-cleansed" data set (NASA93dem) than the cleaner data set (COC81dem). We conjecture that seemingly "dirty" data may contain data that is insightful in some *context*s. While outliers confuse regression-based methods (that fit one model over all data), instance-based tools like $\mathcal{W}2$ can exploit those less-common instances (since they build local models around each *context*).

5.2 What methods for generating project recommendations are hardest to apply?

SEESAW is a more complicated system to implement than $\mathcal{W}2$ since the latter just needs the code of Figure 7 while the former needs an AI search engine *as well as* a software process model. Also, compared to other data miners, $\mathcal{W}2$ is a relatively simpler procedure:

- $\mathcal{W}2$ does not use an intricate recursive descent as done in iterative dichotomization algorithms such as C4.5 or CART (in our experience with teaching data mining, we have found that novices have trouble with that kind of recursion).
- $\mathcal{W}2$ does not use intricate mathematics such as Latent Dirichlet allocation or the quadratic programming used by SVM (in our experience with teaching data mining, we have found that novices rarely code those tools from scratch; rather, novices typically download and apply other people's packages for that work).
- In terms of the learners we have implemented, the only one simpler than $\mathcal{W}2$ are (1) a Naive Bayes classifier on discretized data or (2) a K=1 nearest neighbor. Note that once a developer has access to the source code for a K=1 nearest neighbor, then $\mathcal{W}2$ is less than 100 lines more code (in a high-level programming language like Python).

Not only is SEESAW more complex than $\mathcal{W}2$ to build, it can also add onerous demands to how data is collected from the domain. Like any data farming tool, SEESAW can only process project data in a format compatible with the underlying model. In practice, this limits the scope of the tool. For example, our current version of SEESAW uses Boehm's COCOMO models. Hence, we can only reason about projects expressed in the COCOMO format. Note that $\mathcal{W}2$ does not suffer from such data dependencies. For example, none of the examples studied in §3.3 used the COCOMO structure shown in Figure 5. Rather, the attributes of the examples of §3.3 included the number of basic logical transactions, query count, the number of distinct business units serviced, and several other attributes not referenced in COCOMO.

Finally, SEESAW is harder to maintain than $\mathcal{W}2$. Like any CBR system, $\mathcal{W}2$'s knowledge is instantly updated as soon as the training data is changed. On the other hand, SEESAW is much harder to maintain since its knowledge is held within the intricacies of its model. As an example of how complicated that can be, consider the following:

- We have used SEESAW to offer advice to several projects at NASA's Jet Propulsion Laboratory.
- When business users saw the generated recommendations, they often wanted extra constraints added to the model to handle local policies; e.g. "do not increase automatic tools usage without increasing analyst capability" or "increase the weighting given to the defects term in Equation 1 for high reliability systems".
- After two years of working with SEESAW, the system contained two dozen extra constraints, some with intricate cycle relationships between them.

This is troubling since, while Boehm's original models have been extensively validated [73], we doubt that this certification extends to our modified models.

5.3 Conclusion Reliability and Validity

This paper is a case study that studied the effects of varying some treatments on some data sets. This section discusses limitations of such case studies.

As we read standard texts on empirical software engineering (e.g. Runeson & Höst [74]) the term "case study" often refers to some act which includes observing or changing the conditions under which humans perform some software engineering action. In the age of data mining and model-based optimization this definition should be further extended to include case studies exploring how data collected from different projects can be generalized across multiple projects. As Cruzes et al. comment:

> *Choosing a suitable method for synthesizing evidence across a set of case studies is not straightforward... limited access to raw data may limit the ability to fully understand and synthesize studies.* [75]

This quote from Cruzes et al. is almost a summary of the specific goals of this paper: given limited amounts of software process data, it is best to extrapolate between those data points:

- Use a CPU-intensive data farming method?
- Or to perform some locality-based reasoning using CBR?

Note that it is not possible to explore all the different ways to generalize data collected from individual case studies since:

- There are many methods to perform that generalization[3];
- There is limited time to explore those methods.

That is, a paper that attempts to generalizing across multiple case studies is itself a case study. Hence, any case studies

3. The next section of this paper cites review articles that list hundreds of different optimizers (e.g. [76]). The same large range of options exist for other data mining tools (decision trees, neural nets, genetic algorithms, etc). For example, recently we listed the design decisions made by different researchers using case-based reasoning for effort estimation: when combined together, there are several thousand ways to configure such a CBR tool [77].

in SE data mining or optimization can only explore a small subset of options, selected by the biases of the researcher. For example, the main experiments of this paper compare just two algorithms: SEESAW and $\mathcal{W}2$. Runeson & Höst comment that case studies biased in this way are subject to criticsm:

> *As they are different from analytical and controlled empirical studies, case studies have been criticized for being of less value, impossible to generalize from, being biased by researchers etc.*

For this reason, it is important to state those biases and explore how they affect the *reliability* and *validity* of the stated result.

5.3.1 Reliability

Reliability refers to the consistency of the results obtained from the research. It has at least two components: *internal* and *external* reliability.

Internal reliability checks if an independent researcher reanalyzing the data would come to the same conclusion. Note that large variances in the results threaten internal reliability. For this reason, one of the assessment criteria of this paper was "optimization failure"; i.e. where the treated project data had a larger spread in results than the untreated data. Note that one of the reasons we recommend $\mathcal{W}2$ over data farming is that the latter was observed to exhibit more optimization failures.

External reliability assesses how well independent researchers could reproduce the study. To increase external reliability, this paper has taken care to clearly define our algorithms. Also, all the data used in this work is available on-line in the PROMISE code repository.

5.3.2 Validity

Validity refers to the extent to which a piece of research actually investigates what the researcher purports to investigate. Validity has at least three components: *internal, construct, and external* validity [74].

Internal validity checks that if the differences found in the treatments can be ascribed to the treatments under study. In this paper, the treatments refer to the algorithms built extensively extrapolated the existing data or CBR tools that used small local models. We noted that data farming exhibited a larger number of optimization failures, which we argue as an indication that data farmer over-extrapolates the data to derive inappropriate conclusions, in regions of the data that is sparsely populated. CBR, on the other hand, exhibited far fewer such failures suggesting that nearest neighbor methods that build local "bridges" across local data is safer than the extrapolation used in data farming (at least that is, for sparsely populated data sets).

Construct validity refers to qualities that we cannot directly observe but that we assume they exist in order to explain behavior we can observe. It is extremely important to define constructs that are being investigated in such a way that would otherwise enable an outsider to identify these characteristics if they came across them. In our case, the important construct is the optimization failure. If another data farming researcher is working on data where all their treatments selected from projects with a large variance than the original population, then that should be an indication that they might wish to explore CBR in place of their data farmers.

Finally, *external validity* checks if the results are of relevance for other cases, or can be generalized from samples to populations. For example, the datasets used here comes from COCOMO and NASA projects and these projects do not represent the space of all possible projects. Also, optimization is a large and active field of research (see the range of optimizers discussed in the next section) and any single study can only use a small subset of the known algorithms. To reduce the effects of external validity, this paper first sorted the AI algorithms of Figure 6, then compared the best of that set of algorithms to $\mathcal{W}2$. Nevertheless, it is a important area for future work to observe if other optimizers can produce better results than $\mathcal{W}2$ in data-starved domains.

For case study research, it is difficult to completely avoid problems of external validity The best we can do is to define our algorithms and publicize our data to enable replication of our results by other researchers, and perhaps, point out a previously unknown bias in our analysis. Desirably, other researchers will emulate our methods to repeat, refute, or improve our results.

6 Other Related Work

Some of the related work to this research was discussed above. This section reviews the remaining related work.

6.1 Standard Numeric Optimization

When discussing this work, a frequently asked question is "why use CBR or data farming instead of standard numeric optimizers?". Such optimizers use *gradient descent* that assume an objective function $F(X)$ is differentiable at any point N. A Taylor-series approximation of $F(X)$ decreases fastest if the negative gradient $(-\Delta F(N))$ is followed from point N. An example of the state-of-the-art in gradient descent are the Quasi-Newton (QN) methods (possibly augmented a BFGS update and heuristics for jumping away from discontinuities in the solution when those discontinuities are discovered) [78].

Note that numeric optimizers assume they can access the gradient about a specific point. This, in turn, assumes *coefficient stability*; thus that the slope around some point is known with some certainty. At least for the effort estimation data sets, this assumption is very problematic. For example, consider Boehm's COCOMO parametric model (the 1981 version [72]):

$$Effort = a * Loc^b * \prod_i \beta_i x_i \tag{4}$$

Here x_i is an *effort multiplier* and β_i is a coefficient that controls the influence of x_i (the a, b parameters are the *calibration* parameters).

Our empirical results show that, for effort estimation data sets, we cannot assume stability in the coefficients. To demonstrate, we applied Boehm's method for finding coefficients for the COCOMO attributes [41]. This was repeated twenty times using 66% samples (selected at random) of the NASA93 data set from the PROMISE repository. The results are shown in

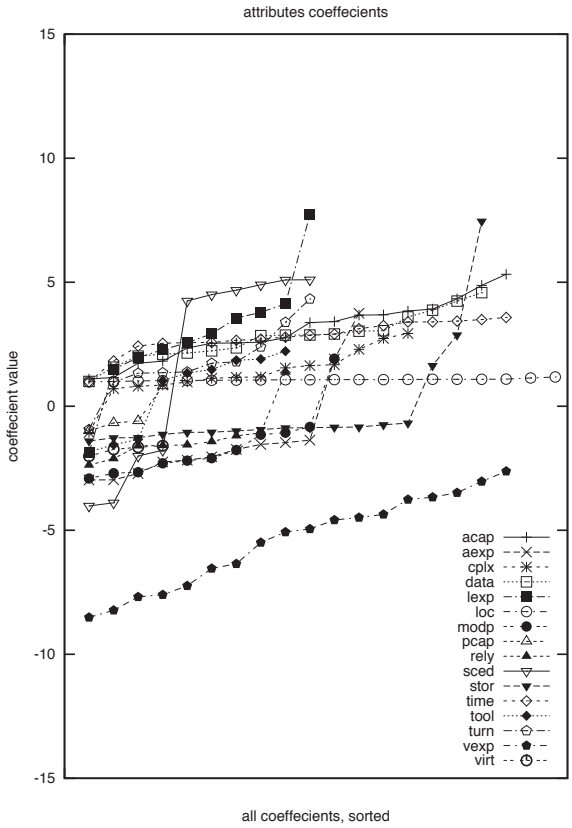

Fig. 12: COCOMO 1 effort multipliers, and the sorted coefficients found by linear regression from twenty 66% sub-samples (selected at random) from the NASA93 PROMISE data set [79]. Prior to learning, training data was *linearized* in the manner recommended by Boehm (x was changed to $log(x)$; for details, see [79]). 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 un-linearized.

Figure 12. While some of the coefficients are stable (e.g. the white circles of *loc* remains stable around 1.1), the coefficients of other attributes are highly unstable:

- The $(max - min)$ range of some of the coefficients is very large; e.g. the upside down black triangles of *stor* ranges from $-2 \leq \beta_i \leq 8$.
- Consequently, nine of the coefficients in Figure 12 jump from negative to positive.

We have seen similar β_i instabilities in other data sets, including Boehm's COCOMO81 data [79]. Accordingly, we cannot recommend standard numeric optimization for learning project management decisions (at least, not use the kinds of process data that we have seen previously).

6.2 Other Optimizers

If standard numeric optimizers are inadequate, then perhaps alternate optimization methods might suffice. When discussing SEESAW, the paper reviewed six such alternate optimizers (see Figure 6) and the literature contains many more:

- Some methods map discrete values true/false into continuous ranges of 1 and 0, and then use *integer programming methods* such as CPLEX [80].
- Other approaches use *meta-heuristic methods* like *tabu search* [81], the simulated annealing method explored in this paper, or the *genetic algorithms* [82]. GAs have been applied to COCOMO data set by many researchers including Sheta [83] and Li et al. [29]. For a review on the state-of-the-art of evolutionary programming and predictive modeling, see Afzal & Torkar [84].

For yet another methods, see the extensive survey of Gu et al. [76] who list hundreds of model-based search methods. No article is able to explore all the above methods but we have some comments on why we do not use some of them.

Any meta-heuristic search method that requires data sampling from the domain is probably inappropriate for data sets as small as those shown in Figure 2. An alternative approach would be to create a model, then use it to generate the data required for a Monte Carlo analysis that is guided by a meta-heuristic search. This is exactly the notion that led us to the model-based search of COCOMO models described above. Note that we no longer recommend that approach since our instance-based methods are performing comparatively better.

Similar to our work, research into search-based software engineering (SBSE) eschews standard numerical optimizers for the exploration of software engineering data. A premise of the SBSE community [85]–[87] is that due to the computational complexity of these problems, exact optimization techniques of operations research like linear programming or dynamic programming are mostly impractical for software engineering problems. Because of this, researchers and practitioners in that field use search technologies to find near-optimal or good-enough solutions. For a review of techniques emerging from the SBSE community that relate to predictions, see Harman [88].

Harman recognizes that Dolado pioneered the use of evolutionary algorithms for software effort estimation [89], [90] (and for more recent work, see [29], [83], [84]). In this respect, this work has some overlap with $\mathcal{W}2$. However, as mentioned above, merely estimating effort is only half the task of $\mathcal{W}2$ (and the other half is finding what to *change* such that some set of quality measures are most altered).

Historically, our work had much overlap to SBSE. Prior to $\mathcal{W}2$, we used a standard SBSE algorithm (simulated annealing) to explore the input space of our models [20]. Based on the results of this paper, we are not currently exploring SBSE for small process data sets. Nevertheless, one branch of SBSE might be of particular interest for $\mathcal{W}2$. Harman [88] reports a class of SBSE algorithms called Bayesian Evolutionary algorithms (BEA). BEA's use evolutionary algorithms to search for models with the highest posterior probabilities. This is a promising approach that could have application to $\mathcal{W}2$'s Bayesian ranking scheme.

6.3 Exploring Other Goals

The experiments demonstrated in this paper focus on *pre-release defect, development time*, and *development effort*. Other quality goals of interest, that are not explored here, are the *post-release development effort* studied by (amongst others) Banker & Slaughter [91], who used a DEA analysis

for that work. DEA indicates on the appropriate weightings to input and output variables. It is a linear programming framework that selects weights that maximize the weighted sum of outputs divided by weighted sum of inputs, subject to constraints on those weightings (in this regard it has some similarities to the feature selection work of, say, Hall & Holmes [92]). Like the Pareto frontier used in evolutionary programming [82], [93], DEA prunes dominated solutions (those that provide less impact on outputs). Incorporating DEA into $\mathcal{W}2$ is a future work under development.

Other kinds of quality goals include minimizing maintenance effort (as studied by Banker & Slaughter) or the release planning problem introduced by Bagnall et al. [94] (explored using evolutionary algorithms by Zhang et al. [95] including the Pareto optimal approach of Saliu & Ruhe [93]). We conjecture that in large multi-requirement projects, $\mathcal{W}2$ could be a tactical tool to make small adjustments to strategic plans generated by other algorithms.

6.4 Parametric and Non-Parametric Methods

For our final notes on related work, we discuss parametric and non-parametric methods for modeling software processes.

$\mathcal{W}2$ is a *non-parametric method* since it makes no assumptions about the underlying distributions of the attribute ranges. Other SE research avoids such parametric assumptions:

- DEA, as discussed in §6.3, does not assume parametric models of any specific format;
- Meta-heuristic search algorithms, like those used in SBSE, are also agnostic about whether or not the models that they are exploring are parametric or non-parametric.

Other research assumes *parametric methods* including methods that with a single underlying parametric function. For example:

- Zhang [96] proposes that software defects follow a Weibull distribution;
- Boehm [72] proposed that development effort was exponentially proportional to lines of code;
- Pendharkar et al. [97] a CobbDouglas function (a convex production function where, for a fixed software size and team size, there exists a unique minimum software development effort).

In theory, we prefer parametric over non-parametric methods. Such parametric methods are better grounded in core principles and their models can be used as summaries of the important effect within certain domains.

However, in practice, many domains lack the data to generate stable settings to the coefficients of the parameter. In those domains, parametric methods can generate unstable conclusions. It turns out that our data sets exhibit coefficient instability (see §6.1). Accordingly, we have tried reducing coefficient instability via:

- Feature subset selection to prune spurious details [98], [99];
- Instance subset selection to prune irrelevancies [61].

Instance subset selection failed to reduce model instability [61]. Feature subset selection has also been disappointing:

- It reduces the performance variance[4] somewhat (in our experiments, from 150% to 53% [99]);
- However the residual median error rates are still large enough that it is hard to use the predictions of these models as evidence for the value of some proposed approach.

After years of unfruitful research on reduced models, feature subset selection, and instance subset selection, we were keen to experiment on some different technology. That leads to the development of $\mathcal{W}2$.

7 Conclusion & Future Work

This paper has compared methods for learning changes to software projects in data-sparse and potentially noisy domains:

1) A small extension to standard CBR (a greedy search over the neighborhood of some query, and then divided into *best* and *rest* regions);
2) A data farmer.

Surprisingly the conclusion is that this relatively simple CBR system out-performed the state-of-the-art data farmer. Note that this argument would not be possible without the comparison with the data farmer. That is, before the simpler approach can be evaluated and endorsed. It may be necessary to spend months/years to certify a much more complex approach. In our study, the result was to discard years of work that recalls a quote from Ken Thompson, one of the inventors of UNIX, who said "one of my most productive days was throwing away 1,000 lines of code" [100]. As researchers, we should always seek for such a simplification, lest our tools grow needlessly complicated[5].

To explain why CBR succeeds, we note that CBR does not attempt to fit a single model across all the data. Instead, it builds multiple micro-models (one for each test instance). In such a way, extraneous influences from remote examples can be largely ignored.

To explain why data farming fails or at least performed less desirable, it is useful to refer to Figure 1:

- For a mathematical audience, we say that the landscape generated by a data farmer is not a clear boundary. Rather, that boundary is itself a distribution which exhibits large variance in regions with low support in the training data set. Any lesson learned in that high variance region has a *low* probability of intersecting the data distributions of the test set. Hence, it has a *high* probability of failing when applied to the test set.
- For a less mathematical audience, we say that the landscape is like a frozen sheet of ice that is thickest in regions of greatest data. Where the training data is sparse, the lessons learned are standing on very thin ice and should not be trusted.

4. By variance, we mean how much each estimate x_i varies from the mean (μ) of all N estimates; specifically $\sum_i^N (x_i - \mu)^2/N$.

5. The late Steve Jobs, founder of Apple Computers, once expressed a similar view: "Simple can be harder than complex: You have to work hard to get your thinking clean to make it simple. But its worth it in the end because once you get there, you can move mountains."

As to external validity, we conjecture that *any* data farming method will likely to make incorrect decisions if it generates elaborated extrapolations of very small software process data sets such as in Figure 2. Very small data sets contain limited amounts of information. Any extrapolation of that information runs into the risk of over-fitting the data and performing poorly on as-yet-unseen test data.

This is not to conclude that *all* data farming tools can *always* be replaced with simpler CBR methods like $\mathcal{W}2$. The test domain of this paper is very specific; i.e. using historical logs of effort/defect data to select minimal sets of most effective changes to a project. The generality of our conclusions must be validated by means of experimentation in other domains. For example, the release planning problem discussed in [94], [95], [101] is a process problem of great complexity. We mentioned above that for such a complex domain, $\mathcal{W}2$ might be the best as a tactical tool to adjust the project structures generated from more complex model-based methods.

Nevertheless, the results of this paper have been greatly piqued our interest in CBR and we plan to further explore this promising technology in the following areas:

1) Are model-based methods worse for noisier data?
2) Is "data cleansing" recommended for regression, but deprecated for instance-based methods?
3) How best to reduce spread, thus increase the confidence a user has in the results?
4) Are there better settings for $\mathcal{W}2$ than those of Figure 7?
5) Can CBR tools like $\mathcal{W}2$ be used for other quality improvement tasks (e.g. minimizing maintenance effort)?
6) Can DEA and Bayesian evolutionary programming improve $\mathcal{W}2$?
7) For small data sets like Figure 2, does CBR always perform better than model-based methods?

REFERENCES

[1] M. Rosenbluth, "Genesis of the monte carlo algorithm for statistical mechanics: a talk at los alamos national laboratory," June 2003.

[2] M. Feather and T. Menzies, "Converging on the optimal attainment of requirements," in *IEEE Joint Conference On Requirements Engineering ICRE'02 and RE'02, 9-13th September, University of Essen, Germany*, 2002, available from http://menzies.us/pdf/02re02.pdf.

[3] S. Islam, J. Keung, K. Lee, and A. Liu, "Empirical prediction models for adaptive resource provisioning in the cloud," *Future Gener. Comput. Syst.*, vol. 28, no. 1, pp. 155–162, Jan. 2012.

[4] V. T. K. Tran, K. Lee, A. Fekete, A. Liu, and J. Keung, "Size estimation of cloud migration projects with cloud migration point (cmp)," in *Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '11, 2011, pp. 265–274.

[5] T. Menzies, O. Elrawas, D. Baker, J. Hihn, and K. Lum, "On the value of stochastic abduction (if you fix everything, you lose fixes for everything else)," in *International Workshop on Living with Uncertainty*, 2007.

[6] T. Menzies and Y. Hu, "Just enough learning (of association rules): The TAR2 treatment learner," in *Artificial Intelligence Review*, 2007, available from http://menzies.us/pdf/07tar2.pdf.

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

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

[9] T. Menzies, S. Williams, O. Elrawas, D. Baker, B. Boehm, J. Hihn, K. Lum, and R. Madachy, "Accurate estimates without local data?" *Software Process Improvement and Practice*, vol. 14, pp. 213–225, 2009.

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

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

[12] O. El-Rawas and T. Menzies, "A second look at faster, better, cheaper," *Innovations in Systems and Software Engineering*, pp. 319–335, 2010.

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

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

[15] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic review of fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, no. PrePrints, 2011.

[16] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Transactions on Software Engineering*, January 2007, available from http://menzies.us/pdf/06learnPredict.pdf.

[17] E. Kocaguneli, T. Menzies, A. Bener, and J. Keung, "Exploiting the essential assumptions of analogy-based effort estimation," *IEEE Transactions on Software Engineering*, vol. 99, no. PrePrints, 2011.

[18] T. Menzies, A. Butcher, A. Marcus, T. Zimmermann, and D. Cok, "Local vs global models for effort estimation and defect prediction," in *IEEE ASE'11*, 2011, available from http://menzies.us/pdf/11ase.pdf.

[19] T. Menzies, A. Butcher, D. Cok, A. Marcus, L. Layman, F. Shull, B. Turhan, and T. Zimmermann, "Local vs. global lessons for defect prediction and effort estimation," *IEEE Transactions on Software Engineering*, p. 1, 2012, available from http://menzies.us/pdf/12localb.pdf.

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

[21] D. Geletko and T. Menzies, "Model-based software testing via treatment learning," in *IEEE NASE SEW 2003*, 2003, available from http://menzies.us/pdf/03radar.pdf.

[22] G. Dyson, *Turing's Cathedral: The Origins of the Digital Universe*. Pantheon, 2012.

[23] N. Silver, *The Signal and the Noise: Why So Many Predictions Fail - But Some Don't*. Penguin, 2012.

[24] G. Horne and T. Meyer, "Data farming: discovering surprise," in *Simulation Conference, 2004. Proceedings of the 2004 Winter*, vol. 1, dec. 2004.

[25] N. E. Fenton, *Software Metrics*. Chapman and Hall, London, 1991.

[26] E. Mendes, I. D. Watson, C. Triggs, N. Mosley, and S. Counsell, "A comparative study of cost estimation models for web hypermedia applications," *Empirical Software Engineering*, 2003, 8(2):163-196.

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

[28] D. Baker, "A hybrid approach to expert and model-based effort estimation," Master's thesis, Lane Department of Computer Science and Electrical Engineering, West Virginia University, 2007.

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

[30] D. Strickland, P. McDonald, and C. Wildman, "Nostromo: Using monte carlo simulation to model uncertainty risk in cocomo ii," in *18th COCOMO Forum, Los Angeles, http://goo.gl/SpB9K*, 2003.

[31] I. Myrtveit, E. Stensrud, and M. Shepperd, "Reliability and validity in comparative studies of software prediction models," *IEEE Trans. Softw. Eng.*, vol. 31, no. 5, pp. 380–391, May 2005.

[32] M. Shepperd and G. F. Kadoda, "Comparing software prediction techniques using simulation," *IEEE Trans. Software Eng*, vol. 27, no. 11, pp. 1014–1022, 2001.

[33] D. Pearce, "The induction of fault diagnosis systems from qualitative models," in *Proc. AAAI-88*, 1988.

[34] A. van Lamsweerde and E. Letier, "Integrating obstacles in goal-driven requirements engineering," in *Proceedings of the 20th International Conference on Software Engineering*. IEEE Computer Society Press, 1998, pp. 53–62, available from http://citeseer.nj.nec.com/vanlamsweerde98integrating.html.

[35] L. Chung, B. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, 2000.

[36] E. Chiang and T. Menzies, "Simulations for very early lifecycle quality evaluations," *Software Process: Improvement and Practice*, vol. 7, no.

3-4, pp. 141–159, 2003, available from http://menzies.us/pdf/03spip.pdf.

[37] W. Heaven and E. Leiter, "Simulating and optimising design decisions in quantitative goal models," in *The 19th IEEE International Conference on Requirements Engineering*, 2011, pp. 79–88.

[38] D. Rodriguez, M. Ruiz, J. C. Riquelme, and R. Harrison, "Multiobjective simulation optimisation in software project management," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, ser. GECCO '11, 2011, pp. 1883–1890.

[39] K. Deb and D. Kalyanmoy, *Multi-Objective Optimization Using Evolutionary Algorithms*. New York, NY, USA: John Wiley & Sons, Inc., 2001.

[40] Y. Jiang, M. Li, and Z.-H. Zhou, "Mining extremely small data sets with application to software reuse," *Software—Practice Experience*, vol. 39, no. 4, pp. 1–27, 2009. [Online]. Available: http://portal.acm.org/citation.cfm?id=1527071

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

[42] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, 1983, number 220, 4589:671-680.

[43] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *J. Chem. Phys*, vol. 21, pp. 1087–1092, 1953.

[44] S. Craw, D. Sleeman, R. Boswell, and L. Carbonara, "Is knowledge refinement different from theory revision?" in *ECML*, 1994, pp. 32–34.

[45] B. Selman, H. A. Kautz, and B. Cohen, "Local search strategies for satisfiability testing," in *Proceedings of the Second DIMACS Challange on Cliques, Coloring, and Satisfiability*, 1993.

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

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

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

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

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

[51] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," *IEEE Trans. on Software Engineering*, vol. 23, no. 12, November 1997.

[52] G. Kadoda, M. Cartwright, L. Chen, and M. Shepperd, "Experiences using case-based reasoning to predict software project effort," in *EASE*, 2000, pp. 23–28.

[53] J. Li and G. Ruhe, "Analysis of attribute weighting heuristics for analogy-based software effort estimation method aqua+," *ESEM*, vol. 13, pp. 63–96, February 2008.

[54] J. Li and R. Ruhe, "A comparative study of attribute weighting heuristics for effort estimation by analogy," in *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, ser. ISESE '06. New York, NY, USA: ACM, 2006, pp. 66–74. [Online]. Available: http://doi.acm.org/10.1145/1159733.1159746

[55] J. Li and G. Ruhe, "Decision support analysis for software effort estimation by analogy," in *PROMISE*, 2007, p. 6.

[56] Y. Li, M. Xie, and G. T., "A study of the non-linear adjustment for analogy based software cost estimation," *Empirical Software Engineering*, pp. 603–643, 2009.

[57] J. Keung, "Empirical evaluation of analogy-x for software cost estimation," in *ESEM*. New York, NY, USA: ACM, 2008, pp. 294–296.

[58] J. Keung, B. A. Kitchenham, and D. R. Jeffery, "Analogy-x: Providing statistical inference to analogy-based software cost estimation," *IEEE Trans. Softw. Eng.*, vol. 34, no. 4, pp. 471–484, 2008.

[59] J. Keung and B. Kitchenham, "Experiments with analogy-x for software cost estimation," in *ASWEC '08*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 229–238.

[60] C. Kirsopp and M. Shepperd, "Case and feature subset selection in case-based software project effort prediction," in *Proc. 22nd SGAI Int'l Conf. Knowledge-Based Systems and Applied Artificial Intelligence*, 2002.

[61] E. Kocaguneli, G. Gay, T. Menzies, Y. Yang, and J. W. Keung, "When to use data from other projects for effort estimation," in *IEEE ASE*, 2010.

[62] P. Pendharkar and J. Rodger, "The relationship between software development team size and software development cost," *Communications of the ACM*, vol. 52, no. 1, pp. 141–144, 2009.

[63] D. K. D.W. Aha and M. Albert, "Instance-based learning algorithms," in *Machine Learning*, 1991, pp. 37–66.

[64] Y. Yang and G. Webb, "Weighted proportional k-interval discretization for naive-bayes classifiers," in *Proceedings of the 7th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2003.

[65] R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufman, 1992, iSBN: 1558602380.

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

[67] G. Gay, T. Menzies, O. Jalali, G. Mundy, B. Gilkerson, M. Feather, and J. Kiper, "Finding robust solutions in requirements models," *ASE*, vol. 17, no. 1, pp. 87–116, 2010.

[68] G. Gay, T. Menzies, M. Davies, and K. Gundy-Burlet, "Automatically finding the control variables for complex system behaviour," *Automated Software Engineering*, no. 4, December 2010.

[69] W. Cohen, "Fast effective rule induction," in *ICML*, 1995, pp. 115–123.

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

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

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

[73] S. Chulani, B. Boehm, and B. Steece, "Bayesian analysis of empirical software engineering cost models," *IEEE Trans. on Software Engineering*, 1999, 25(4), 1999.

[74] P. Runeson and M. Host, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, pp. 131–164, 2009.

[75] P. R. M. H. Daniela Cruzes, Tore Dyba, "Case studies synthesis: Brief experience and challenges for the future," in *ESEM'11: the International Symposium on Empirical Software Engineering and Measurement*, 2011.

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

[77] E. Kocaguneli, T. Menzies, A. Bener, and J. Keung, "Exploiting the essential assumptions of analogy-based effort estimation," *IEEE Transactions on Software Engineering*, vol. 28, pp. 425–438, 2012, available from http://menzies.us/pdf/11teak.pdf.

[78] C. Sims, "Matlab optimization software," QM&RBC Codes, Quantitative Macroeconomics & Real Business Cycles, Mar. 1999.

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

[80] H. Mittelmann, "Recent benchmarks of optimization software," in *22nd Euorpean Conference on Operational Research*, 2007.

[81] F. Glover and M. Laguna, "Tabu search," in *Modern Heuristic Techniques for Combinatorial Problems*, C. Reeves, Ed. Oxford, England: Blackwell Scientific Publishing, 1993.

[82] D. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[83] A. F. Sheta, "Estimation of the cocomo model parameters using genetic algorithms for nasa software projects," *Journal of Computer Science*, vol. 2, no. 2, pp. 118–123, 2006.

[84] W. Afzal and R. Torkar, "Review: On the application of genetic programming for software engineering predictive modeling: A systematic review," *Expert Syst. Appl.*, vol. 38, pp. 11 984–11 997, September 2011.

[85] M. Harman and B. Jones, "Search-based software engineering," *Journal of Info. and Software Tech.*, vol. 43, pp. 833–839, December 2001.

[86] J. Clarke, J. Dolado, M. Harman, R. Hierons, B. Jones, M. Lumkin, B. Mitchell, S. Mancoridis, K. Rees, M. Roper, and M. Shepperd, "Reformulating software engineering as a search problem," *Software, IEE Proceedings* -, vol. 150, no. 3, pp. 161 – 175, june 2003.

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

[88] M. Harman, "The relationship between search based software engineering and predictive modeling," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, ser. PROMISE '10. New York, NY, USA: ACM, 2010, pp. 1:1–1:13. [Online]. Available: http://doi.acm.org/10.1145/1868328.1868330

[89] J. Dolado and L. Fernandez, "Genetic programming, neural networks and linear regression in software project estimation," in *INSPIRE*, 1998.

[90] J. J. Dolado, "A validation of the component-based method for software size estimation," *IEEE Transactions of Software Engineering*, vol. 26, no. 10, pp. 1006–1021, 2000.

[91] R. D. Banker and S. A. Slaughter, "A field study of scale economies in software maintenance," *Management Science*, vol. 43, no. 21, pp. 1709–1725, December 1997.
[92] M. Hall and G. Holmes, "Benchmarking attribute selection techniques for discrete class data mining," *IEEE Transactions On Knowledge And Data Engineering*, vol. 15, no. 6, pp. 1437– 1447, 2003.
[93] M. O. Saliu and G. Ruhe, "Bi-objective release planning for evolving software systems," in *ESEC/FSE*. Dubrovnik, Croatia: ACM, 3-7 September 2007, pp. 105–114.
[94] A. Bagnall, V. Rayward-Smith, and I. Whittley, "The next release problem," *Information and Software Technology*, vol. 43, no. 14, December 2001.
[95] H. Zhang and X. Zhang, "Comments on 'data mining static code attributes to learn defect predictors'," *IEEE Trans. on Software Engineering*, September 2007.
[96] H. Zhang, "On the distribution of software faults," *Software Engineering, IEEE Transactions on*, vol. 34, no. 2, pp. 301 –302, march-april 2008.
[97] G. S. P.C. Pendharkar, J .A. Rodger, "An empirical study of the cobb–douglas production function properties of software development effort," *Information and Software Technology*, pp. 1181–1188, 2008.
[98] Z. Chen, T. Menzies, and D. Port, "Feature subset selection can improve software cost estimation," in *PROMISE'05*, 2005.
[99] T. Menies, K. Lum, and J. Hihn, "The deviance problem in effort estimation," in *PROMISE, 2006*, 2006.
[100] E. S. Raymond, *The Art Of Unix Programming*. Addison-Wesley, 2003.
[101] A. Ngo-The and G. Ruhe, "Optimized resource allocation for software release planning," *IEEE Trans. on Software Engineering*, 2009, 35(1):109-123.

Tim Menzies (Ph.D, UNSW) is a full Professor in CS at WVU; and the author of 200+ referred papers. He is an associate editor of IEEE TSE and the journals of Empirical Software Engineering journal, and the Automated Software Engineering. He co-founded the PROMISE conference on reproducible experiments in SE (see http://promisedata.googlecode.com). Web site http://menzies.us; vita: http://goo.gl/8eNhY; publications: http://goo.gl/8KPKA.

Adam Brady is a master's student at West Virginia University. His current research focuses on instance-based reasoning, specifically its application to improving software quality.

Jacky Keung (Ph.D, UNSW) is an Assistant Professor in CS, City University, Hong Kong. He was a Research Scientist in SE Research Group NICTA. He also holds an academic position, CSE, UNSW. An active software engineering researcher, his main research interests are in software cost estimation, empirical modeling and evaluation of complex systems, and data-intensive analysis for SE data and its application to project management.

Jairus Hihn (Ph.D. U.Maryland) is a principal member of the engineering staff at JPL, CalTech and manager for the Software Quality Improvement Projects Measurement Estimation and Analysis Element. He has built estimation models and providing software and mission level cost estimation support to JPLs Deep Space Network and flight projects since 1988. He has extensive experience in in the areas of decision analysis, institutional change, R&D project selection cost modeling, and process models.

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

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

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

Barry Boehm (Ph.D. UCLA) served from 1989 and 1992 as Director of DARPA's Information Science and Technology Office, and as Director of the DDR&E Software and Computer Technology Office. He worked at TRW from 1973 to 1989, finally as Defense Systems Chief Scientist. Before that we worked at the Rand Corporation from 1959 to 1973, finally as Head of the IS Department. His research explores software process modeling, software requirements and architectures and metrics & cost models, SE environments, and knowledge-based SE.