Comparing Case-Based Reasoning with Model-Based Systems for Software Effort Estimation

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

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

ABSTRACT

Background: Machine learning has been incredibly successful. We are drowning in choice as to how we can infer the cost of building software. Two conflicting choices ask whether we should construct an abstract model to represent software projects, or if simply reasoning from historical cases is enough.
Aim: We aim to show a comparison between two software cost estimation methods, one case-based and the other model-based.
Method: We present $\mathcal{W}$, a case-based, calibrationless, model-agnostic reasoning algorithm, and compare its software cost reduction estimations to those of SEESAW, a model-based method for local AI search within COCOMO.
Results: In 32 different tests $\mathcal{W}$ always performs at least as good as SEESAW. For 9 of those tests $\mathcal{W}$ performs better.
Conclusion: $\mathcal{W}$ performs just as well if not better than SEESAW for reducing software cost across multiple goals such as development effort, project defects, and completion time.

Keywords

Software Effort Estimation, Cost Models, Case Based Reasoning

1. INTRODUCTION

Accurately estimating software cost remains a core aspect of businesses today. For example, in the case of NASA's Check-out Launch Control System [22], the initial estimate of \$200 million was quickly overrun, with the project burning another \$200 million before being cancelled. Not only are these estimates often wrong [2], they are often wrong by a factor of four or more [8].

To assist humans in creating these estimates, a torrent of machine learning research has emerged. A quick search over major journals and conferences revealed over 600 different papers on estimating software cost. [18]. Of these papers, the dominant topic was to introduce new estimation techniques. What users are left with is an abundance of choice. Without a single, unifying banner to weave together cost estimation techniques, no one method has achieved *de facto* status in the cost estimate community.

For example, the standard method of software cost estimation involves parametric model construction. A representation of a software project is generalized into a mathematical abstraction, then all future projects are estimated according to the parameters of the model. A benefit of which is that once a model can show general trends, extrapolate patterns, and generally reduce complex, human behavior into a mere formula.

There several reasons for the lack of a standard software cost model. These includes generality, data islands and instability. Software models may not be general so it can be inappropriate to apply a software model learned in environment one to environment two. Also, many companies prefer to keep cost related data confidential. This data island effect has also contributed to the fragmentation of the field by preferring building private models rather than using publicly available models. This multiplicity of software effort models has lead to scarcity of the publicly available local data needed for the model based effort estimation. Without sufficient data to build, audit, and tune models, the predictions generated by these models may be highly unstable. [15]

However, the locality of software models may not necessarily be a bad thing. Rather than attempt to generalize all software into a standard model, we can instead simply reason from the data. Case-based reasoning attempts to capitalize on two tenents of the world: similar problems have similar solutions, and that problems encountered once are likely to occur again. [9] As C. Riesbeck explains, "We don't think, we remember." [16]

Model-based and case-based methods represent two ends of a wide spectrum of cost estimation techniques. Rather than attempt to settle an ongoing debate, we simply offer an experiment between the two strategies. This paper introduces $\mathcal{W}$, a simple, case-based algorithm for software effort estimation and reduction. Competing with $\mathcal{W}$ is SEESAW, a model-based (USC COCOMO [2]) algorithm that extends from MaxWalkSat. [7] $\mathcal{W}$ is a much smaller program with hundreds of lines of awk, whereas SEESAW exists as thousands of lines of C++/LISP. The relative complexity of these two algorithms resides not just in their languages; we will show the intrinsic simplicity of $\mathcal{W}$ compared to the Monte Carlo simulator an AI search of SEESAW.

This paper is structured in the following order: First we present a deeper look at the technical tradeoffs of model-based reasoning and case-based reasoning. Second, we pose three research questions to be answered by the results of our experiment. Then, we'll explore the implementations of SEESAW and $\mathcal{W}$. Finally, we present an experiment that compares the software cost recommendations from $\mathcal{W}$ with those of SEESAW.

1.1 Research Questions

RQ1: Can the recommendations produced by a case based reasoning system such as W perfom better than a model-based approach?

RQ2: Do different datasets effect the relative performance of one method over another?

RQ3: In which situations should one prefer a particular method over another?

2. BACKGROUND

2.1 Estimation Techniques

Sheppherd categorizes the current landscape of software cost estimation methods into three groups. [18]

- Human-centric techniques (a.k.a. expert judgment)
- Algorithmic/parametric models such as COCOMO [2] [3]
- Induced prediction systems such as case-based reasoning.

Human centric techniques are the most widely-used estimation method [18], but are problematic. If an estimate is disputed, it can be difficult to reconcile competing human intuitions (e.g.) when one estimate is generated by a manager who is senior to the other estimator. Also, Jorgensen [5] reports that humans are surprisingly poor at reflecting and improving on their expert judgments. For the purposes of this paper, we will set aside the comparision to human-centric techniques as another topic.

2.2 Modeling Techniques

One alternative to expert judgment is a *model-based* estimate. Models are a reproducible methods for generating an estimate. This is needed for (e.g.) U.S. government software contracts that require a model-based estimate at each project milestone. Such models are used to generate and audit an estimate, or to double-check a human-centric estimate. Model based estimates can be generated using an algorithmic/parametric approach or via induced prediction systems. In the former, an expert proposes a general model, then domain data is used to tune that model to specific projects. For example, Boehm's 1981 COCOMO model [2] hypothesized that development effort was exponential on LOC and linear on 15 effort multipliers such as analyst capability, product complexity, etc. Boehm defined a local calibration procedure to tune the COCOMO model to local data. Induced prediction systems are useful if the available local training data does not conform to the requirements of a pre-defined algorithmic/parametric model such as COCOMO. There are many induction methods including stepwise regression, linear regression, ruleinduction, neural nets, model trees, and analogy, just to name a few. All these methods have underlying assumptions.

2.3 Case Based Reasoning

Case based reasoning is a method of machine learning that seeks to emulate human recollection and adaptation of past experiences in order to find solutions to current problems. That is, as humans we tend to base our decisions not on complex reductive analysis, but on an instantaneous survey of past experiences [17]; i.e. we don't think, we remember. CBR is purely based on this direct adaptation of previous cases based on the similarity of those cases with the current situation. Having said that, a CBR based system has no dedicated world model logic, rather that model is expressedthrough the available past cases in the case cache. This cache is continuously updated and appended with additional cases.

Aamodt & Plaza [1] describe a 4-step general CBR cycle, which consists of:

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

Having verified the results from our chosen adapted action on the new case, the new case is added to the available case base. The last step allows CBR to effectively learn from new experiences. In this manner, a CBR system is able to automatically maintain itself. As discussed below, $\mathcal{W}$ supports *retreive*, *reuse*, and *revise* (as well as *retain* if the user collecting data so decides).

This 4-stage cyclical CBR process is sometimes referred to as the R4 model [21]. Shepperd [21] considered the new problem as a case that comprises two parts. There is a description part and a solution part forming the basic data structure of the system. The description part is normally a vector of features that describe the case state at the point at which the problem is posed.The solution part describes the solution for the specific problem (the problem description part).

The similarity between the target case and each case in the case base is determined by a similarity measure. Different methods of measuring similarity have been proposed for different measurement contexts. A similarity measure is measuring the closeness or the distance between two objects in an n-dimensional Euclidean space, the result is usually presented in a distance matrix (similarity matrix) identifying the similarity among all cases in the dataset. Although there are other different distance metrics available for dif-

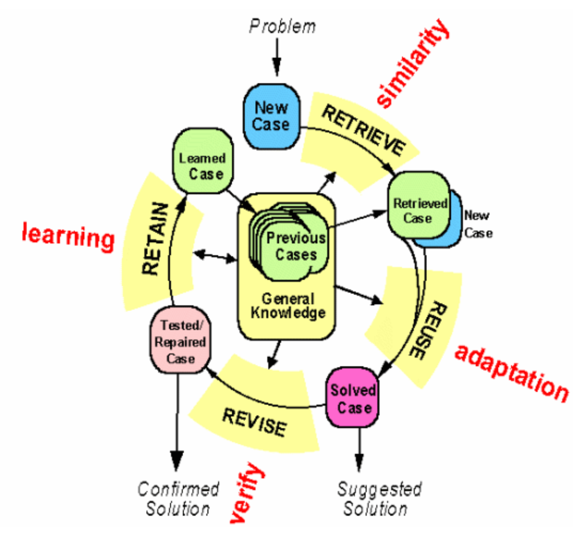

Figure 1: figure
A diagram describing the steps of CBR (source: http://www.peerscience.com/Assets/cbrcycle1.gif).

ferent purposes, the Euclidean distance metric is probably the most commonly used in CBR for its distance measures.

Irrespective of the similarity measure used, the objective is to rank similar cases from case-base to the target case and utilize the known solution of the nearest k-cases. The value of k in this case has been the subject of debate [6,20]: Shepperd [20], Mendes [11] argue for $k = 3$ while Li [10] propose $k = 5$.

Once the actual value of the target case is available it can be reviewed and retained in the case-base for future reference. Stored cases must be maintained over time to prevent information irrelevancy and inconsistency. This is a typical case of incremental learning in an organization utilizing the techniques of CBR.

Observe that these 4 general CBR application steps (retreive, reuse, revise, retain) do not include any explicit model based calculations; rather we are relying on our past experience, expressed through the case base, to estimate any model calculations based on the similarity to the cases being used. This has two advantages:

1. It allows us to operate independently of the models being used. For example, our prior report to this conference [4] ran over two data sets. This this study, based on CBR, uses twice as many data sets.
2. This improves our performance, since data retrieval can be more efficient than calculation, especially given that many thousands of iterations of calculation were needed with our traditional modeling based tool. As evidence of this, despite the use of a slower language, $\mathcal{W}$'s AWK code runs faster than the C++/LISP used in SEESAW. It takes just minutes to conduct 20 trials over 13 data sets with $\mathcal{W}$. A similar trial, conducted with SEESAW, can take hours to run.

3. IMPLEMENTATIONS

3.1 SEESAW

SEESAW searches within the ranges of project attributes to find constraints that most reduce development effort, development time, and defects. Figure 2 shows SEESAW's pseudo-code. The code is an adaption of Kautz & Selman's MaxWalkSat local search procedure [13]. The main changes are that each solution is scored via a Monte Carlo procedure (see score in Figure 2) and that SEESAWseeks to minimize that score (since, for our models it is some combination of defects, development effort, and development time in months).

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

3.2 W

The standard procedure for CBR is to report the median class value of some local neighborhood. This neighborhood is typically defined as the Euclidean distance from a defined project in n-dimensional space with n project features. [20] $\mathcal{W}$ works similarly, but defines a project as a range of values:

- From a range of project values, cases are retrieved that match a specific amount of *overlap* with the defined project ranges. A case's overlap is defined as the percentage of attributes that fall within the specified ranges of the defined project.
- From these selected similar cases, the cases are sorted by a measure of utility. In the case of effort, defect, and month estimations, this utility is the normalized euclidean distance from the lowest possible cost for all three factors.
- From these sorted ranges, a *contrast set* is learned. The top 5 "best" cases (those with the best utility measure) are placed into a set labelled "best". The next 15 ranked cases are placed into a set labelled "rest", for a combined total of 20 cases.

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

Figure 2: Pseudocode for SEESAW

- From the contrast set, $\mathcal{W}$ selects the features that best select for the region with the best utility measurements.

3.2.1 Contrast Sets

Once a contrast set learner is available, it is a simple matter to add $\mathcal{W}$ to CBR. $\mathcal{W}$ finds contrast sets using a greedy search, where candidate contrast sets are ranked by the frequency of which they appear in the "best" set squared divided by how often the candidate appears in both the "best" and "rest" sets. A simple strategy to score more favorably towards attributes that occur most often in the best case is to square the number of times. Taking this heuristic one step further, given an attribute x, we can penalize x's occurance in the "rest" by dividing the sum of the frequency counts in best and rest [13], the ensuring rare attributes are weighted appropriately:

$$like = \frac{freq(x|best)^2}{freq(x|best) + freq(x|rest)} \tag{1}$$

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

3.2.2 The W Algorithm

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

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

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

Figure 3: $\mathcal{W}$'s syntax for describing the input query q. Here, all the values run 1 to 6. $4 \leq cplx \leq 6$ denotes projects with above average complexity. Question marks denote what can be controlled- in this case, $rely, time$ (required reliability and development time)

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

1. *Retrieve*: The initial query q is used to find the N training cases nearest to q using a Euclidean distance measure where all the attribute values are normalized from 0 to 1.
2. *Reuse* (adapt): The N cases are sorted by effort and divided into the K_1 best cases (with lowest efforts) and K_2 rest cases. For this study, we used $K_1 = 5, K_2 = 15$. Then we seek the contrast sets that select for the K_1 best cases with lowest estimates. All the attribute ranges that the user has marked as "controllable" are scored and sorted by LOR. This sorted order S defines a set of candidate q' queries that use the first *i-th* entries in S:
$$q'_i = q \cup S_1 \cup S_2 ... \cup S_i$$

Formally, the goal of $\mathcal{W}$ is find the smallest i value such q'_i selects cases with the least median estimates.

According to Figure 1, after *retrieving* and *reusing* comes *revising* (this is the "verify" step). When revising q', $\mathcal{W}$ prunes away irrelevant ranges as follows:

1. Set $i = 0$ and $q'_i = q$
2. Let $Found_i$ be the test cases consistent with q'_i (i.e. that do not contradict any of the attribute ranges in q'_i).
3. Let $Effort_i$ be the median efforts seen in $Found_i$.
4. If $Found$ is too small then terminate (due to overfitting). After Shepperd [20], we terminated for $|Found| < 3$.
5. If $i > 1$ and $Effort_i < Effort_{i-1}$, then terminate (due to no improvement).

6. Print q_i' and $Effort_i$.
7. Set $i = i + 1$ and $q_i' = q_{i-1} \cup S_i$
8. Go to step 2.

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

3.3 Multiple Goals

Optimizing for simply effort in software cost estimation can be bad. For instance, it may require less effort to produce software with much lower quality standards, but doing so may drastically increase defects. Additionally, reducing the schedule constraints will incur larger development time. To solve this problem, $\mathcal{W}$ bases it's recommendations on multiple goals.

It does this by ranking on some measure of utility for each case. If, say, we wish to reduce the effort, defects, and months requirements, a simple utility measure can be derived from the normalized euclidian distance from the best possible values for all goals (**E**ffort, **D**efects, **M**onths):

$$score = \frac{E - E_{min}}{E_{max} - E_{min}} + \frac{D - D_{min}}{D_{max} - D_{min}} + \frac{M - M_{min}}{M_{max} - M_{min}}$$

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

4.1 Methodology

We present an experiment to compare the effort, defects, and months reduction power of W compared to SEESAW. Each algorithm will be given a set of 4 real-world projects with defined COCOMO attribute ranges. The goal of each algorithm will be to choose specific recommendations for these value ranges that will maximize the reduction the project's effort, defects, and months.

Before we can measure the relative effectiveness of one effort reduction recommendation over another, we need a way to compare the recommendations from both SEESAW and W in common terms. This is trivial, as once each tool generates its recommendations, these recommendations can then be applied using any model or inference prediction system. Once applied, regardless of the testing environment we can compare the relative effectiveness of each algorithm. For the sake of simplicity we have chosen to test recommendations using W.

4.2 Dataset and Project Descriptions

In order to test the efficacy of model-based and case-based estimation techniques, we need a source of case data for W. Because SEESAW was built around COCOMO and W is a data-agnostic, we chose the free COCOMO NASA93 dataset available from `http://promisedata.org/data`. This dataset represents 93 different NASA projects collected from the 1980's and 1990's represented as feature vectors describing each project in COCOMO format.

However, the NASA93 data only contains historical information for actual project effort. In order to provide a comparison in reducing multiple goals, the development time in

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

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

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

months was computed using the COCOMO model. Also, the COQUALMO [3] model for defect removal was employed to synthesize *defects* estimations for each project. Combined, each historical project case had three attributes that needed to be minimized.

Next, project definitions are needed to provide grounds on which to improve. Along with the NASA93 dataset comes four real-life case studies at NASA, each representing a unique range of possible COCOMO scale factors and effort multipliers.

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

Figure 4 shows in detail each project. The low and high ranges define the space of possible recommendations for that project. For instance, the reliability of the JPL flight software can vary from a ranking of 3 (nominal) to 5 (very high). SEESAW and W will each attempt to improve the overall estimation in effort, defects, and months by recommending changes to these values.

4.3 Testing Procedure

A testing rig was built to inject recommendations generated within SEESAW into W's testing phase. Because the testing phase of W only requires the recommendations gained from the training phase, testing with SEESAW's recommendations is identical to testing with W's.

Once in place, recommendations for each of the four case studies was tested 50 times. Each test reported the reduction in median effort, defects, months, and the synthesized "score" attribute.

This measure gives a more general picture of the overall reductions across the entire space rather than each individual goal.

Also reported is the "spread", or the distance between the third (75%) and first (25%) quartiles. This measure gives a rough estimate on how much more *certain* the estimations have become. A reduction in this "spread" constitutes a narrowing of the range of estimations, implying a less varied outcome. Both the median reduction and the spread reduction are reported in terms of before and after estimates.

5. RESULTS

The four datasets were split into four separate goals (score, effort, defects, months) measured as both average reduction (lower cost) and average "spread" reduction (narrower recommendations). The following figures show the results for "score"[12, 13], effort[6, 7], defects[8, 9], and months[10, 11].

To determine statistical significance for each comparison, A Mann Whitney U test was performed on the two sets of reduction distributions from each comparison. Given a 95% confidence interval, figure 5 shows that overall W performs better than SEESAW on 9 of the 32 tests, and performs no worse than SEESAW on the other 21 tests.

On individual datasets, W performs better on reducing estimations given projects with more constrained ranges such as OSP2. On 5 out of the 8 comparisons involving OSP2, W outperformed SEESAW. Less conclusive results were achieved regarding the other, more openly-defined projects. Only 3 out of the 24 comparisons resulted in a statistically significant improvement using *W* over SEESAW.

Overall, *W* performs no worse than SEESAW, and shows clear improvements in certain circumstances.

6. ANSWERS TO RESEARCH QUESTIONS

With the above results, we can now answer the research questions proposed at the beginning of the paper.

RQ1: Can the recommendations produced by a case based reasoning system such as W perfom better than a model-based approach?

Algorithm	Wins	Losses	Ties
W	9	0	21
SEESAW	0	9	21

Figure 5: Win/Loss/Tie table for statistically significant reductions across all goals with the Nasa Flight, Ground, OSP, and OSP2 projects.

Average Median Effort Reductions

Dataset	Algorithm	Before	After
Ground	SEESAW	269	197
	W	269	184
Flight	SEESAW	258	252
	W	258	208
OSP	SEESAW	270	195
	W	270	210
OSP2	SEESAW	291	269
	W	291	227

Figure 6: Average Median Effort Reductions for the NASA93 Dataset (Wins highlighted)

Average Effort "Spread" Reductions

Dataset	Algorithm	Before	After
Ground	SEESAW	526	99
	W	526	125
Flight	SEESAW	567	204
	W	567	165
OSP	SEESAW	350	114
	W	350	100
OSP2	SEESAW	824	418
	W	824	299

Figure 7: Average Median Effort Reductions for the NASA93 Dataset (Wins highlighted)

Average Median Defect Reductions

Dataset	Algorithm	Before	After
Ground	SEESAW	2666	2107
	W	2666	2035
Flight	SEESAW	2812	3138
	W	2812	3017
OSP	SEESAW	3180	2688
	W	3180	2867
OSP2	SEESAW	2612	7797
	W	2612	2271

Figure 8: Average Median Defect Reductions for the NASA93 Dataset (Wins highlighted)

Average Defect "Spread" Reductions

Dataset	Algorithm	Before	After
Ground	SEESAW	5979	1670
	W	5979	1784
Flight	SEESAW	6292	3457
	W	6292	2347
OSP	SEESAW	5250	2949
	W	5250	2011
OSP2	SEESAW	6527	6216
	W	6527	3777

Figure 9: Average Median Defect Reductions for the NASA93 Dataset (Wins highlighted)

Average Median Month Reductions

Dataset	Algorithm	Before	After
Ground	SEESAW	16.7	13.5
	W	16.7	13.7
Flight	SEESAW	16.0	13.7
	W	16.0	14.5
OSP	SEESAW	16.2	14.4
	W	16.2	14.2
OSP2	SEESAW	16.5	14.6
	W	16.5	14.2

Figure 10: Average Median Month Reductions for the NASA93 Dataset (Wins highlighted)

Average Month "Spread" Reductions

Dataset	Algorithm	Before	After
Ground	SEESAW	10.5	7.3
	W	10.5	4.1
Flight	SEESAW	10.3	9.4
	W	10.3	7.0
OSP	SEESAW	10.1	6.8
	W	10.1	5.9
OSP2	SEESAW	11.8	7.2
	W	11.8	6.8

Figure 11: Average Median Month Reductions for the NASA93 Dataset (Wins highlighted)

Average Median "Score" Reductions

Dataset	Algorithm	Before	After
Ground	SEESAW	0.186	0.129
	W	0.186	0.130
Flight	SEESAW	0.171	0.139
	W	0.171	0.146
OSP	SEESAW	0.166	0.125
	W	0.166	0.131
OSP2	SEESAW	0.165	0.160
	W	0.165	0.129

Figure 12: Average Median "Score" Reductions for the NASA93 Dataset (Wins highlighted)

Average "Score" "Spread" Reductions

Dataset	Algorithm	Before	After
Ground	SEESAW	0.282	0.200
	W	0.282	0.200
Flight	SEESAW	0.287	0.235
	W	0.287	0.233
OSP	SEESAW	0.272	0.215
	W	0.272	0.195
OSP2	SEESAW	0.290	0.264
	W	0.290	0.225

Figure 13: Average Median "Score" Reductions for the NASA93 Dataset (Wins highlighted)

We've demonstrated that $\mathcal{W}$ can perform just as well if not better than SEESAW 5. More importantly, similar results have been achieved while making fewer assumptions, and without the need for any model calibration.

RQ2: Do different datasets effect the relative performance of one method over another?

The results for the OSP2 dataset[12, 13, 6, 7, 8, 9, 10, 11] show that there exists a discrepancy between performance in datasets with few recommendation options.

RQ3: In which situations should one prefer a particular method over another?

This is left as an open question. One of the ongoing issues in effort estimation is model vs case based. While we currently cannot extrapolate enough evidence to make a solid statement in the general case, we offer the preceeding comparison as a nexus point for further research along with a few considerations.

One consideration is how much local data is available. In situations where, say, a company has recently formed or is breaking new ground in software engineering practices, the lack of relevant historical cases prevents the use of CBR. As Martin Shepphard says, "the use of accurate, systematic, historical data for building useful effort prediction systems is extremely important, yet in practice, such data is seldom available." [19] However, previous work has shown that one may not need much data in order to successfully perform CBR tasks [12]. After all, "history repeats itself, but not exactly" [18].

Another consideration is how satisfying a justification is from concrete, documented cases compared to abstract mathematical extrapolations? In our experience, users, especially business users, have found more solace in being able to use their own intuition to justify a recommendation. The core algorithms inside $\mathcal{W}$ such as contrast sets are fast and easy to explain, bearing one accepts the underlying locality assumption.

Finally, one must consider the lack of a single, unifying model for software effort estimation. Building a model requires assumptions from the underlying statistical distributions of the data to the calibration parameters needed to ensure accuracy. While a model benefits from reuse once created, as you only need to build it once, the potential range of these parameters [15] can undermine the usefulness of the tool as a whole.

We haven't faulted previous work [4] [14] with model-based approaches, but we have demonstrated that tools like $\mathcal{W}$ can compete and exceed more complex methods. With only a few extensions to the premise of case based reasoning, $\mathcal{W}$ has shown promise.

6.1 Threats to Validity

Currently, we do not have results for datasets with actual historical measurements for defects and months. Synthesizing the defects and months for the NASA93 dataset using COQUALMO and COCOMO may have removed underlying patterns in the data. Because these values were generated from COCOMO scale factors and effort multipliers, a direct correlation exists between defects and months in regards to the project features.

Secondly, the comparison between SEESAW and $\mathcal{W}$ took place inside $\mathcal{W}$'s testing procedure. While this decision allowed a direct comparision between recommendations, the estimations generated for each were just that, estimations. We do not have a means to compare how these recommendations might perform in a real-world setting at this time.

7. CONCLUSION

$\mathcal{W}$ has shown tangible potential in terms of direct performance, but the question of choosing model-based reasoning over case-based reasoning remains an ongoing discussion. Research scientists relish the idea that we can reduce complex, human-centered domains to beautiful, intuitive patterns. However, while the models we produce may be useful, effort estimation has yet to find a single, unifying abstraction under which to explain all software projects. As long as this goal remains unrealized, tradeoffs will exist between model-based reasoning and case-based.

8. REFERENCES

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

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

[3] 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.

[4] P. Green, T. Menzies, S. Williams, and O. El-waras. Understanding the value of software engineering technologies. In *IEEE ASE'09*, 2009. Available from `http://menzies.us/pdf/09value.pdf`.

[5] M. Jorgensen and K. Molokken-Ostvold. Reasons for software effort estimation error: Impact of respondent error, information collection approach, and data analysis method. *IEEE Transactions on Software Engineering*, 30(12), December 2004.

[6] G. Kadoda, M. Cartwright, L. Chen, and M. Shepperd. Experiences using casebased reasoning to predict software project effort, 2000.

[7] H. Kautz, B. Selman, and Y. Jiang. A general stochastic approach to solving problems with hard and soft constraints. In D. Gu, J. Du, and P. Pardalos, editors, *The Satisfiability Problem: Theory and Applications, New York, NY*, pages 573–586, 1997. Available on-line at `http://citeseer.ist.psu.edu/168907.html`.

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

[9] D. Leake. *Case-based reasoning: experiences, lessons, and future directions*. AAAI Press, Menlo Park, CA, USA, 1996.

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

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

[12] T. Menies, K. Lum, and J. Hihn. The deviance problem in effort estimation. In *PROMISE, 2006*, 2006. Available from `http://menzies.us/06deviations.pdf`.

[13] T. Menzies, O. El-Rawas, J. Hihn, and B. Boehm. Can we build software faster and better and cheaper? In *PROMISE'09*, 2009. Available from `http://menzies.us/pdf/09bfc.pdf`.

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

[15] T. Menzies, S. Williams, O. Elrawas, D. Baker, B. Boehm, J. Hihn, K. Lum, and R. Madachy. Accurate estimates without local data? *Software Process Improvement and Practice*, 14:213–225, July 2009. Available from `http://menzies.us/pdf/09nodata.pdf`.

[16] C. Riesbeck. *What next? The future of case-Based Reasoning in Post-Modern AI*. AAAI Press, 1996.

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

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

[19] M. Shepperd and M. Cartwright. Predicting with sparse data. *IEEE Transactions on Software Engineering*, 27:987–998, 2001.

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

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

[22] Spareref.com. Nasa to shut down checkout & launch control system, August 26, 2002. `http://www.spaceref.com/news/viewnews.html?id=475`.