From Effort Estimation to Effort Reduction

Adam Brady[1], Tim Menzies[1], Jacky Keung[2]
[1]*Lane Department of CS&EE, West Virginia University, USA,*
[2]*NICTA, Australia,*
adam.m.brady@gmail.com,tim@menzies.us,Jacky.Keung@nicta.com.au

***Abstract*—A major hurdle still faced by data mining practitioners involves translating complex theories into actionable items for managers. As theories and methodology becomes more complex, business users must place more faith in the theory assumptions rather than their own intuition. To combat this we present an extremely simple effort reduction recommendation system called "W" that maintains no underlying parametric model and produces simple, concise theories complete with explanations. "W" does this by extending the effort estimation of case-based reasoning with the explanation power of contrast sets. From these contrast sets we build new queries that constrain the original effort estimates, demonstrating a reduction in both the median effort and variance of historically-relevant projects.**

***Keywords*-software economics; artificial intelligence; case-based reasoning; CBR by analogy; explanation; effort reduction; human-in-the-loop learning**

I. Introduction

"Don't tell me where I am, tell me where to go."
-A very busy user

A machine learner's purpose in life is to generate theories. However, all theories must eventually be read by humans. Therefore, we should consider what kinds of theories people like to read.

If the reader is a busy person they might not need or even be able to use the nuance of a complex theory. Rather, a busy person might instead just want to know what will bring the most benefit from the least effort and nothing more. It follows that machine learning for busy people should not strive for elaborate theories or increasing the expressive power of the theory's language. Rather, a better goal might be to find the smallest theory with the most impact.

Previous work explored minimal controllers in the context of AI search over monte-carlo analysis of software process models using STAR [1]. A concern with that line of research is that the more elaborate our tech, the harder it is to validate the conclusion. In particular, reproducability is hard when it takes 2-6 months to rebuild experimental rig discussed in the previo us paper. Is there a simpler way to generate simple most-effective controllers for a software project? Another problem was the conclusions were dependent on the parametric model used. For example, our previous work was based on the USC COCOMO tools. The tools were built from mostly southern-caifornia DoD aerospace contractors and it is a valid criticism of those models that the external validity of those models is an open question.

Therefore, in order to simplify the reproducibility of our work and remove our dependency of possibly inappropriate models, this paper reports an experiment in learning controllers using case-based reasoning. We report a remarkably simple CBR algorithm dubbed "W" that achieves all our goals. "W" generates recommendations that reduce both the median effort and the variance on the effort. The recommendations are very small in the case-studies shown here, they're usually one change to a project with widely varying recommendations between projects.

II. Background

A. Modeling vs. Case-Based Reasoning by Analogy

Much work has been put forth in developing parametric models for estimation software effort. Popular models such as USC COCOMO [2] take the reductionist approach of reducing any arbitrary software project to a handful of descriptive parameters. Each parameter consists of a range of values from 1 to 6 representing ranges from very low to extremely high, respectively, mapped to regression values expressing how each attribute affects effort. The result of which can be used to model software effort using this formula:

$$effort = a * KSLOC^{b+0.01*\sum_{i=1}^{5} SF_i} * (\prod_{j=1}^{17} EM_j) \quad (1)$$

B. The Case for CBR

Where $KSLOC$ represents thousands of lines of code, SF represents exponential scale factor attributes, and EM represents linear effort multipliers. While these parameters were derived from historical NASA projects and have shown their utility, an effort estimator must rely that these parameters are reflective of any given arbitrary project. As time progresses, one can speculate that the relevance of the underlying model will remain applicable to future cases. Also, such models often require careful domain-specific tuning, as is the case of the a and b linear and exponential tuning values in 1. For the original NASA dataset [3], these values can range anywhere from:

$$(2.2 \leq a \leq 9.18) \wedge (0.88 \leq b \leq 1.09) \quad (2)$$

Because these tunings are arbitrary, one can see how merely the tuning parameter a can quadruple the effort estimation depending on its value. Given this uncertainty and reliance on establishing a general case for all software projects, another method, case-based reasoning, establishes an alternative methodology for effort estimation.

The general form of case-based reasoning differs from parametric model-based learning in that there is no attempt to reduce complex phenomena to a general-case parametric model. Rather, historical data is *Retrieved, Reused, Revised, and Retained* [4] (the "4 REs") as a means of adapting the past to reason about the future.

The intuition of CBR comes from cognitive models of human behavior. That is, as humans we tend to base our decisions not on complex reductive analysis, but on an instantaneous survey of past experiences. In other words, we don't think, we remember. For example one not need consider the actuarial ramifications of a kitchen grease fire, our past experiences tells us without hesitance: Fire hot, fire bad. While the power of simulating and modeling the world has allowed for our success as a species, our abilities fall apart without historical knowledge of the world.

However, for this paper we are more interested in a subset of the CBR domain, effort estimation by analogy.

Analogy effort estimation centers around historical project data frozen in time, referred to as $cases$, to reason about new project instances. The core assumption being that given some measurement of relevancy between cases and incoming project instances, we can generate a query q over the historical space that selects for the kth nearest neighbors , or *knn*, most relevant to a project instance. Because the cases are analogous to the project instance being considered, CBR methodology states we can reason about the project instance by examining the analogous cases.

It is important to note, unlike established models CBR does carry the requirement of obtaining relevant and accurate historial data that represents the problem space. While this can limit the deployable situations for CBR, when this data *is* available CBR offers an intrinsic justification of it's estimations that are less obvious with a parametric approach. One need only examine the analogous cases for evidence supporting a particular estimation.

Finally, an added benefit of reasoning from historical data grants CBR distinct advantages over parametric models as the learner is only exposed to conditions that happen in practice, rather than reason about all possible theoretical possibilities. This is useful when presented with data containing outlier behavior that might not map to a linear regression as easily. As long as the data collection requirements are met, CBR can be applied to any consistent and representative means of describing a software project.

For these reasons outlined above we include CBR by analogy as the foundation of our learning controller, "W," as a starting point in generating useful, actionable, simple theories.

C. From Estimation to Planning

Upon a thorough review of CBR literature, there exists little work beyond estimation. Famous CBR systems such as Shepperd's ANGEL [5] and more recent work such as Keung's Analogy-X [6] focus their efforts on quantifying what is relevant in effort estimation, but remain limited in the scope of simply estimating. While means of deciding what cases are relevant is a fundamental problem of CBR, it is worth persuing other possible applications of this unique methodology. For instance, no where does there exist any planning in CBR literature.

Our focus of this paper and "W" is not to explore the effort estimation power of learning by analogy. Rather, we seek to broaden the scope of CBR to include generating theories for effort reduction. To do this, we must devise a way to exploit the knowledge gained from the attributes of the relevant neighborhood of projects, rather than merely their historical effort.

Conceptually, if we query the historical space for a set of relevant cases, we should recommend project changes that cause our project instance to be more similar to historical cases with the most favorable outcome. In the case of effort reduction, a good recommendation would drive our project away from attributes associated with worse effort outcomes and towards projects with better effort outcomes. When tested against unseen cases, a project with the previously recommended attribute constraints should demonstrate a reduction in both the effort and variance seen in its neighbors. From these test cases we can build successive queries, q_i^*, that incrementally apply these attribute constraints until we can no longer accurately predict a reduction on effort:

$$query_i^* = query + \cup_i c_i \tag{3}$$

Each attribute constraint c_i can then be added to the query q_i in descending order of recommendation score.

III. Deciding with "W"

"W" is our implementation of a effort reduction planning system using the fundamentals of CBR. "W" consists of three main steps:

- Defining a Project Instance
- Deciding what attributes are relevant to improving the project
- Evaluating the effectiveness of improvements on test cases

"W" decides based on two core assumptions: Similar projects have similar efforts, and the the attributes that drive effort reduction will occur more frequently in the best cases than the worst.

"W" Project File Example

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

Figure 1. Example project file for "W"

"W" Historical Cases

```
@relation NASA93
@attribute rely 1 2 3 4 5
@attribute data 2 3 4 5
@attribute cplx 1 2 3 4 5 6
@attribute time 3 4 5 6
@attribute stor 3 4 5 6
@attribute virt 2 3 4 5
@attribute turn 2 3 4 5
@attribute acap 1 2 3 4 5
@attribute aexp 1 2 3 4 5
@attribute pcap 1 2 3 4 5
@attribute vexp 1 2 3 4
@attribute lexp 1 2 3 4
@attribute modp 1 2 3 4 5
@attribute tool 1 2 3 4 5
@attribute sced 1 2 3 4 5
@attribute ksloc 8.4 10.8 24 25.2 31.2
36 72 117.6 360
@data
4 2 4 3 3 2 2 3 3 3 3 4 4 3 2 24.6 117.6
4 2 4 3 3 2 2 3 3 3 3 4 4 3 2 7.7 31.2
4 2 4 3 3 2 2 3 3 3 3 4 4 3 2 8.2 36
4 2 4 3 3 2 2 3 3 3 3 4 4 3 2 9.7 25.2
4 2 4 3 3 2 2 3 3 3 3 4 4 3 2 2.2 8.4
4 2 4 3 3 2 2 3 3 3 3 4 4 3 2 3.5 10.8
4 2 4 3 3 2 2 3 3 3 3 4 4 3 2 66.6 352.8
3 2 4 3 3 2 2 4 5 5 3 4 3 3 3 20 72
3 2 4 3 3 2 2 4 5 4 3 4 3 3 3 6 24
3 2 4 3 3 2 2 4 5 5 3 4 3 3 3 100 360
```

Figure 2. Example datafile for "W" consisting of a sample of NASA93 data

	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		
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

Figure 3. Real-world project description from a NASA Orbital Space Plane

A. *Defining Projects in "W"*

Before a we can define a potential, "W" must define its historical cases. Figure 2 demonstrates a small example of how any arbitrary historical data is defined for use by "W." The first line declares the name of the dataset, followed by all attribute (column) names and possible values. Because of this enumeration, currently "W" only supports discretized datasets. After the @data sigil, historical cases are defined numerically assuming the same column order as the attributes defined above.

The main interface between a user and "W" consists of it's project descriptions. A simple example file is given in figure 1. When a project lead or manager wishes to know what can be done to improve the effort of a future project, "W" should only make recommendations that fall within actionable, feasible changes. For example, the historical projects in figure 3 define the COCOMO attribute ranges and hard values that constrain the potential recommendations for the project. In the case of OSP, it is outside the bounds of reality to attempt to change the precedenedness (prec) of a project if the project *has no precedent.*

To define an attribute range as controllable, the "?" character denotes this mutability. For example, in the example project in figure 1 the reliability of this project can be set anywhere from three to five, but for managerial reasons tool use is locked at two. If an attribute is defined but lacks a "?," when generating samples for the knn calculations these attribute ranges will be used but not recommened.

B. *Finding Relevant Cases*

Once a project is defined, "W" by default generates fifty random projects consistent with the attribute constraints defined. This stoicastic sampling solves the problem of finding the nearest neighbors within a range. Using figure 4 as a simplified example, you can see how the attribute ranges defined in the project files allow for a "zone" of potential projects to exist. In the diagram the x dimension represents a continuum of schedule pressure attributes while the y dimension represents a range of managerial experience. For any case, a specific value for both of this attributes exists, as represented by the arrows. Samples generated will always fall within the bounds of the dotted "controllable zone."

From these samples, the k nearest neighbors are determined by calculating the minimum n-dimensional euclidean distance between any case c and generated sample s:

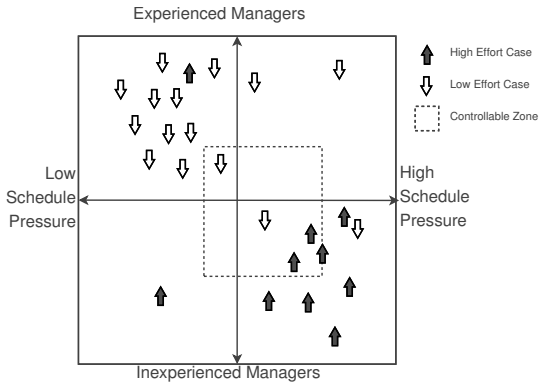

Figure 4. A simplified example of historical cases (arrows) and their relation to the defined project instance ranges (dotted box)

$$dist(c,s) = \sqrt{(c_1 - s_1)^2 + (c_2 - s_2)^2 + \ldots + (c_n - s_n)^2} \tag{4}$$

From these distances, the kth smallest distances are chosen as relevant examples. In the case of "W" by default the 20 nearest neighbors are chosen. These 20 nearest neighbors reprsent our query q of the historical space.

procedure FINDKNN($Samples$,$Cases$)
  for all sample in samples **do**
    for all case in cases **do**
      $dist = euclidianDist(sample, case)$
      $dist = normalize$(dist)
      if distance[case] && distance[case] ¡ dist **then**
        $distance[case] = dist$
      end if
    end for
  end for
  relevant = topK(distance)
end procedure

C. Contrasting The Neighborhood

Now that relevant case examples to our project instance have been chosen, we can consider what causes similar projects to present varying effort measurements. One method presented by Menzies and Hu [7] involves building a contrast set between attributes, separating cases into best and rest then measuring which attributes occur more freqently in best than rest, where the "best" consists of the top k1 neighbors. By default "W" considers the top five of twenty cases to be considered "best" and the other fifteen the "rest."

From this dichotomy, "W" implements two such measurement tools for scoring attributes most likely to occur in best but not rest, Nomograms and the B-squared measure. Both allow for determining which attributes show the largest correlation with lower effort.

1) B-squared: 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 [1], the ensuring rare attributes are weighted appropriately:

$$like = \frac{freq(x|best)^2}{freq(x|best) + freq(x|rest)} \tag{5}$$

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

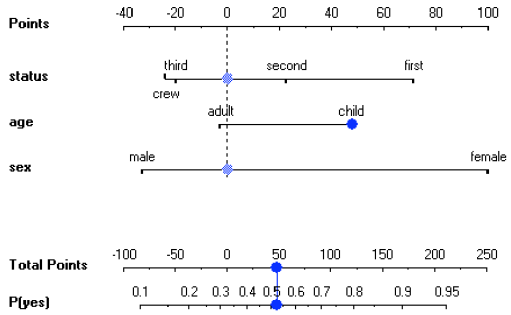

Figure 5. Nomogram modeling survival odds on the HMS Titanic [8]

2) Nomograms: Nomograms use the simple Naive Bayesian rule to determine whether a given attribute will belong to the instance of the "best" neighbors. By taking the log of the odds ratio of an attribute appearing in best compared to rest, we are given an individual score for that attribute. That score allows us to determine what the probability of a particular attribute or combination of attributes occuring in best or rest. By taking the log, these scores have the unique property of obeying the property of addition.

In figure III-C2 the nomograms for survival rates of the RMS Titanic passengers are presented. Each bar represents a different attribute of a passenger, and the labelled marks along that bar represent the attribute values. By adding up the numeric values seen above each attribute value, you can determine how strongly each attribute value predicts for an ourcome of survival. After adding these "points" together, you can calculate the Bayesean probability of survival by converting "Total Points" to P(yes). As shown in figure III-C2, when considering no other attributes a child has a survival chance of slightly above 50%.

For "W," nomograms represent another means to rank attributes with the future potential to combine attribute scores for ranking paired attributes.

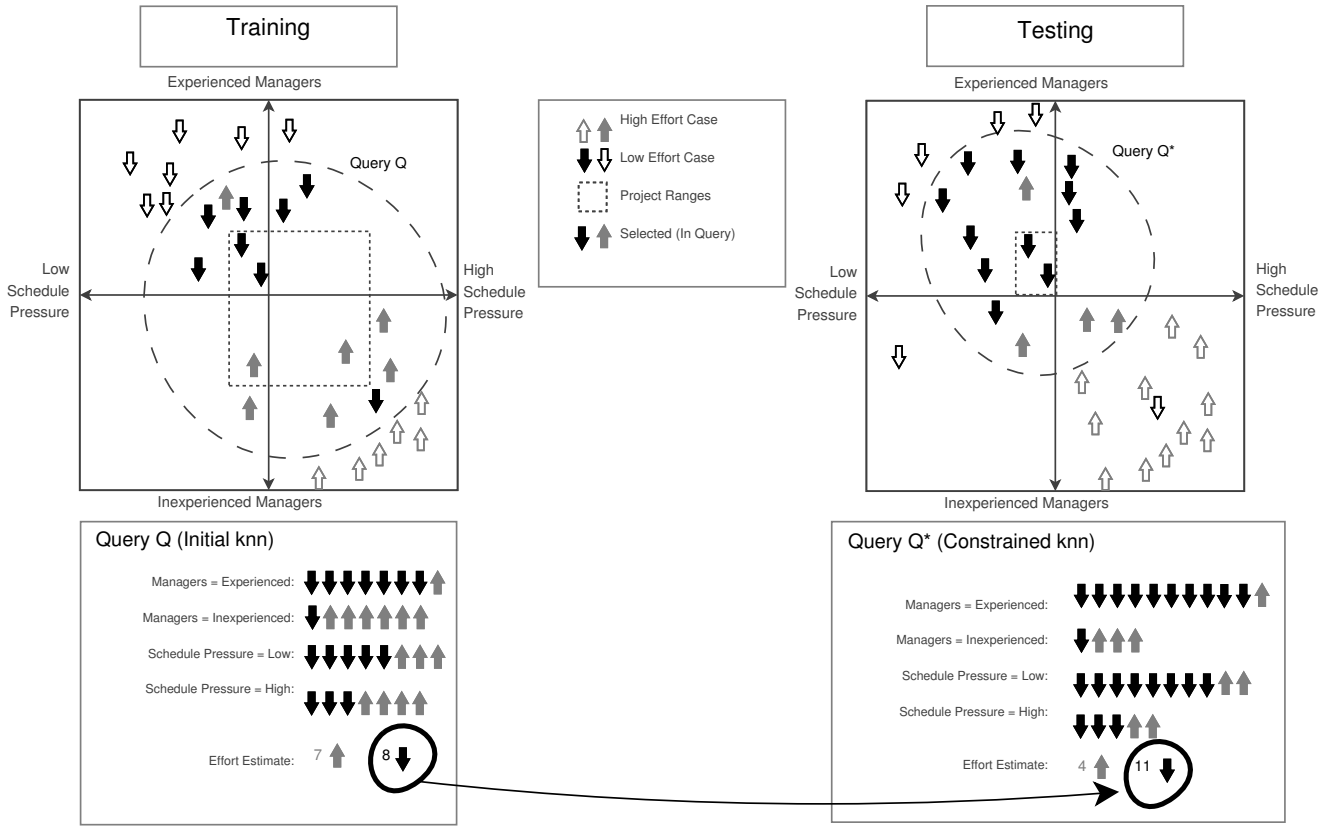

Figure 6. Overview of "W"

D. Generating Plans for Effort Reduction

Now that we've determined what attribute values are important in the query q for reducing effort, we can apply these attributes to an unseen testing set of data to determine whether we've managed to reduce the effort estimation. To do this, we take q and build successive queries on test, incrementally constraining q_i with the highest scoring attribute/value from either the Nomograms or B-squared test.

$$query_i^* = query + \cup_i c_i \tag{6}$$

Each query will reduce the size of the testing set until

As all positively-scoring attribute constraints are applied to q_i^*, depending on the available cases a decision must be made when to stop constraining the theory. While additional treatments may further reduce median effort and variance, additional constraints reduce the size of each query, reducing the historical support for the theory.

Currently, there is future work to be done on developing an automatic stopping rule for early termination of applying treatments. Figure 7 demonstrates an example run of "W." Currently, careful human analysis is recommended to obtain the best treatment from "W" as one can bias a theory towards real world actionable cost or potentially greater median effort and variance reductions. In the example run from the NASA93 dataset, applying the first constraint, $stor = 5$, results in a median reduction of effort from 360 to 170, a reduction of over half. However, this constraint maintains a query with the same maximum effort value as the origial q. The application of the next constraint, $time = 5$, may require additional resources, but the addition of this constraint historically precludes the case with an effort of 4560.

IV. Results and Analysis

An experiment was conducted with "W" across a variety of historical Software Engineering datasets from the PROMISE dataset repository. While the datasets use COCOMO's Lines of Code (KSLOC) and Function Points for quantifying project complexity, any representative discrete model can be used. The datasets used were:

- NASA: effort prediction data representing 93 different NASA projects collected in the 80s and 90s. (COCOMO)
- Desharnais: University of Montreal software costs (Function Points)
- ISBSG: StandAlone and Client/Server software costs (Function Points)

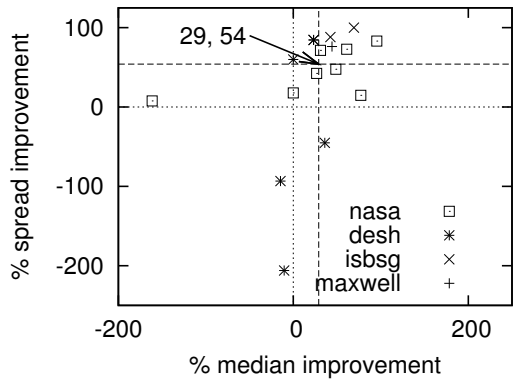

Figure 8. Median effort and median spread reduction across multiple datasets.

Theories Generated by the NASA93 Dataset

Project Dataset	Effort Reduction	Spread Reduction	Theory
coc81 flight	0%	18%	data=3
coc81 ground	95%	83%	pmat=2
coc81 osp	-160%	8%	cplx=5
coc81 osp2	31%	71%	sced=1
nasa93 flight	61%	73%	rely=3
nasa93 ground	77%	15%	aexp=5
nasa93 osp	49%	48%	sced=3
nasa93 osp2	27%	42	sced=3

Figure 9. Example theories generated for NASA projects, demonstrating preliminary evidence for very small, local treatments.

- Maxwell: Finish banking data [9]

For each dataset "W" generated 50 synthetic examples, took the 20 kth nearest neighbors, and ranked the "best" of these neighbors as being in the 5 lowest effort rankings. "W" trained it's theories on 67% of each dataset and applied learned treatments to the other 33%. Nomograms were used as a means of ranking treatments.

As shown in figure 8, the upper-right quadrant of the chart consist of projects where "W" generated theories that reduced both the median effort and th "spread" betweeen the 25% and 75% quartiles. On average we show a 29% reduction in median effort along with a 54% reduction in spread.

While we have consistent improvements across all datasets, outliers in the Desharnais dataset present some problems. This can possibly be attributed to only having synthetic project descriptions rather than real-world representations. For the NASA datasets using USC COCOMO we see larger improvements with the majority of projects demonstrating improvements in both effort and spread reduction. Also, when a decision from "W" negatively impacts either effort or spread reduction, only one of these performance metrics is significantly impacted. There exist no decisions that significantly increase both the spread and effort of our results.

Finally, figure 9 contrasts the different treatments generated within the same dataset. Three interesting patterns emerge. First, the theories we generate are extremely small. For all projects in the NASA93 only a single treatment brings us the largest reduction. Second, each theory is vastly different. Using the USC COCOMO attributes no consistent pattern emerges in our recommendations, despite consistent effort reduction. Finally, the majority of theories generated include values in the middle of the possible ranges.

V. Future Work

An inherit problem with case-based reasoning involves collecting enough data for a significant representation. We are always collecting data, and for "W" there will always be a need for larger, more relevant datasets. Also, there exist multiple schools of thought as to whether lines of code or function points offer a better measure of software complexity. With a greater varity of datasets we may be able to offer a better comparison of these metrics.

The theories generated in figure 9 present an interesting case in explanation stability. Although we consistently reduced median effort and spread, the treatments that did so show no correlation in their ability to do so. Such a result points to a potential lack of generality in software engineering, or a deeper pattern in "W"'s internal decision mechanisms. Given the stance of CBR as a sort of counterpoint to the search for a general theory of software cost, the potential for a strong statement on a lack of generality may bolster the position of CBR research.

Also, given the power of contrast sets to extract relevent attributes, future work exists using "W" as a feature subset selector. Nomograms have proven to be a reliable bayesian means of ranking features, as shown in our effort reductions. Using this knowledge to weight the euclidean distance measure may prove a valueable tool in more accurately defining relevance of historical cases.

VI. Discussion

With "W" we've demonstrated how contrast sets can be used to extend case-based reasoning by analogy into the treatment realm. Even with the simplicity of euclidian nearest-neighbors for measuring case relevancy we achieve significant and consistent software effort reduction(figure 8). We've demonstrated that even small, local theories may offer meaningful improvement for the business user (figure 9).

While an automatic stopping procedure is useful, business users may appreciate the ability for a human expert to sit in-the-loop with "W" and shape treatment options. This flexibility allows a user to easily and quickly decide themselves what theory works best for their situation, and immediately judge the effects of such a theory with historical backing.

These theories are small, easy to understand, and useful for the busy reader. Compared to complex decision trees or rule learners "W" offers a refreshing simplicity with real results.

REFERENCES

[1] 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.

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

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

[4] R. LOPEZ DE MANTARAS, D. MCSHERRY, D. BRIDGE, D. LEAKE, B. SMYTH, S. CRAW, B. FALTINGS, M. L. MAHER, M. T. COX, K. FORBUS, M. KEANE, A. AAMODT, and I. WATSON, "Retrieval, reuse, revision and retention in case-based reasoning," *The Knowledge Engineering Review*, vol. 20, no. 03, pp. 215–240, 2005. [Online]. Available: http://journals.cambridge.org/action/displayAbstract?fromPage=online&aid=435263&fulltextType=RA&fileId=S0269888906000646

[5] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," *IEEE Transactions on Software Engineering*, vol. 23, no. 12, November 1997, available from http://www.utdallas.edu/~rbanker/SE_XII.pdf.

[6] J. Keung and B. Kitchenham, "Experiments with analogy-x for software cost estimation," in *ASWEC '08: Proceedings of the 19th Australian Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 229–238.

[7] T. Menzies and Y. Hu, "Data mining for very busy people," in *IEEE Computer*, November 2003, available from http://menzies.us/pdf/03tar2.pdf.

[8] M. Možina, J. Demšar, M. Kattan, and B. Zupan, "Nomograms for visualization of naive bayesian classifier," in *PKDD '04: Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*. New York, NY, USA: Springer-Verlag New York, Inc., 2004, pp. 337–348.

[9] K. D. Maxwell.

```
historical data : nasa93
new project(s)  : ground
#n      score   range
------  -----   --------------
     1  14.00   stor = 5
     2   1.50   time = 5
     3   1.28   cplx = 5
     4   1.25   acap = 3
     5   0.73   pcap = 3
     6   0.56   rely = 4
     7   0.48   data = 3
     8   0.41   data = 2
     9   0.10   pmat =
    10   0.10   ltex =
    11   0.10   apex =
    12   0.00   stor = 3
    13  -0.06   time = 3
    14  -0.29   rely = 3
    15  -0.43   cplx = 4
    16  -0.69   acap = 4

60 70 72 82 90 107 170 210 215 360 370 409 458 480 571 750 973 1248 2460 4560
                                  25%     50%     75%
           Baseline:              90      360     571

Results of applying the  top n-th ranges found during training

    n=1: stor= (5) :             70      170     480      Tie    5*{70 82 170 480 4560}
and n=2: time= (5) :             70      82      170      Tie    4*{70 82 170 480}
and n=3: cplx= (5) :                     480     480      Tie    1*{480}
```

Figure 7. An example run of "W"

	Definition	Low-end = {1,2}	Medium ={3,4}	High-end= {5,6}
Defect removal features				
execution-based testing	all procedures and tools used for testing	none	basic testing at unit/ integration/ systems level; basic test data management	advanced test oracles, assertion checking, model-based testing
automated analysis	e.g. code analyzers, consistency and traceability checkers, etc	syntax checking with compiler	Compiler extensions for static code analysis, Basic requirements and design consistency, traceability checking.	formalized specification and verification, model checking, symbolic execution, pre/post condition checks
peer reviews	all peer group review activities	none	well-defined sequence of preparation, informal assignment of reviewer roles, minimal follow-up	formal roles plus extensive review checklists/ root cause analysis, continual reviews, statistical process control, user involvement integrated with life cycle
Scale factors:				
flex	development flexibility	development process rigorously defined	some guidelines, which can be relaxed	only general goals defined
pmat	process maturity	CMM level 1	CMM level 3	CMM level 5
prec	precedentedness	we have never built this kind of software before	somewhat new	thoroughly familiar
resl	architecture or risk resolution	few interfaces defined or few risks eliminated	most interfaces defined or most risks eliminated	all interfaces defined or all risks eliminated
team	team cohesion	very difficult interactions	basically co-operative	seamless interactions
Effort multipliers				
acap	analyst capability	worst 35%	35% - 90%	best 10%
aexp	applications experience	2 months	1 year	6 years
cplx	product complexity	e.g. simple read/write statements	e.g. use of simple interface widgets	e.g. performance-critical embedded systems
data	database size (DB bytes/SLOC)	10	100	1000
docu	documentation	many life-cycle phases not documented		extensive reporting for each life-cycle phase
ltex	language and tool-set experience	2 months	1 year	6 years
pcap	programmer capability	worst 15%	55%	best 10%
pcon	personnel continuity (% turnover per year)	48%	12%	3%
plex	platform experience	2 months	1 year	6 years
pvol	platform volatility ($\frac{frequency\ of\ major\ changes}{frequency\ of\ minor\ changes}$)	$\frac{12\ months}{1\ month}$	$\frac{6\ months}{2\ weeks}$	$\frac{2\ weeks}{2\ days}$
rely	required reliability	errors are slight inconvenience	errors are easily recoverable	errors can risk human life
ruse	required reuse	none	multiple program	multiple product lines
sced	dictated development schedule	deadlines moved to 75% of the original estimate	no change	deadlines moved back to 160% of original estimate
site	multi-site development	some contact: phone, mail	some email	interactive multi-media
stor	required % of available RAM	N/A	50%	95%
time	required % of available CPU	N/A	50%	95%
tool	use of software tools	edit,code,debug		integrated with life cycle

Figure 10. Description of COCOMO Scale Factors and Effort Multipliers