

Exploiting the Essential Assumptions of Analogy-based Effort Estimation

Ekrem Kocaguneli, *Student Member, IEEE*, Tim Menzies, *Member, IEEE*,
Ayse Bener, *Member, IEEE*, and Jacky W. Keung, *Member, IEEE*

Abstract—

Background: There are too many design options for software effort estimators. How can we best explore them all?

Aim: We seek aspects on general principles of effort estimation that can guide the design of effort estimators.

Method: We identified the essential assumption of analogy-based effort estimation: i.e. the immediate neighbors of a project offer stable conclusions about that project. We test that assumption by generating a binary tree of clusters of effort data and comparing the variance of super-trees vs smaller sub-trees.

Results: For ten data sets (from Coc81, Nasa93, Desharnais, Albrecht, ISBSG, and data from Turkish companies), we found: (a) the estimation variance of cluster sub-trees is usually *larger* than that of cluster super-trees; (b) if analogy is restricted to the cluster trees with lower variance then effort estimates have a significantly lower error (measured using MRE, AR and Pred(25) with a Wilcoxon test, 95% confidence, compared to nearest-neighbor methods that use neighborhoods of a fixed size).

Conclusion: Estimation by analogy can be significantly improved by a dynamic selection of nearest neighbors, using only the project data from regions with small variance.

Index Terms—Software Cost Estimation, Analogy, k -NN

1 INTRODUCTION

Software effort estimates are often wrong by a factor of four [1] or even more [2]. As a result, the allocated funds may be inadequate to develop the required project. In the worst case, over-running projects are canceled and the entire development effort is wasted. For example, NASA canceled its incomplete Check-out Launch Control System project after the initial \$200M estimate was exceeded by another \$200M [3].

It is clear that we need better ways to generate project effort estimates. However, it is not clear how to do that. For example, later in this paper we document thousands of variations for analogy-based effort estimation (ABE). Effort estimation is an active area of research [4]–[7] and more variations are constantly being developed. We expect many more variations of ABE, and other effort estimation methods, to appear in the very near future.

Recent publications propose data mining toolkits for automatically exploring this very large (and growing) space of options for generating effort estimates. For example, in 2006, Auer et al. [8] proposed an extensive search to learn the best weights to assign different project features. Also in that year, Menzies et al. [9]’s

COSEEKMO tool explored thousands of combinations of discretizers, data pre-processors, feature subset selectors, and inductive learners. In 2007, Baker proposed an exhaustive search of all possible project features, learners, etc. He concluded that such an exhaustive search was impractical [10].

The premise of this paper is that we can do better than a COSEEKMO-style brute-force search through the space of all variants of effort estimators. Such studies are computationally intensive (the COSEEKMO experiments took two days to terminate). With the ready availability of cheap CPU farms and cloud computing, such CPU-investigations are becoming more feasible. On the other hand, datasets containing historical examples of project effort are typically small¹. In our view, it seems mis-directed to spend days of CPU time just to analyze a few dozen examples. These CPU-intensive searches can generate gigabytes of data. Important general properties of the estimation process might be missed, buried in all that data. As shown below, if we exploit these aspects, we can significantly improve effort estimates.

This paper proposes an alternative to brute-force and heuristic search. According to our *easy path* principle for designing an effort predictor:

Find the situations that confuse estimation. Remove those situations.

(Later in this paper, in §3.2, we will offer a precision definition of “confuse estimation”. For now, we need

- Ekrem Kocaguneli is with the Lane Department of Computer Science and Electrical Engineering, West Virginia University. E-mail: ekocagun@mix.wvu.edu
- Tim Menzies is with the Lane Department of Computer Science and Electrical Engineering, West Virginia University. E-mail: tim@menzies.us
- Ayse Bener is with TRSITM, Ryerson University. E-mail: ayse.bener@ryerson.ca
- Jacky W. Keung is with the Department of Computing at The Hong Kong Polytechnic University. E-mail: jacky.keung@comp.polyu.edu.hk

1. For example, the effort estimation datasets used in Mendes et al. [11], Auer et al. [8], Baker [10], this study, and Li et al. [12] have median size (13,15,31,33,52), respectively.

only say that confused estimates are highly inaccurate).

The easy path is not standard practice, i.e. it is a *heuristic*. Usually, prediction systems are matured by adding mechanisms to handle the harder cases (cases for whom estimation accuracy is lower). For example, the AdaBoost algorithm generates a list of learners, and each learner focuses on the examples that were poorly handled by the one before [13].

Focusing on just the easy cases (cases for whom estimation accuracy is higher) could be problematic. If we only explore the easy cases, we could perform badly on the hard test cases. On the other hand, if the easy path works, it finds short-cuts that simplifies future effort estimation work. Also, it avoids COSEEKMO's brute-force search since, according to this principle, we only explore the options that challenge the essential assumptions of the predictor.

The rest of this paper uses the easy path to build and evaluate an effort estimator called TEAK (short for "Test Essential Assumption Knowledge" and available at <http://unbox.org/wisp/tags/teak/>). In keeping with the easy path, we only explored design options that commented on TEAK's essential assumptions; specifically: (a) case subset selection and (b) how many training examples should be used for estimation.

TEAK's design applied the easy path in five steps:

- 1) Select a prediction system.
- 2) Identify the predictor's essential assumption(s).
- 3) Recognize when those assumption(s) are violated.
- 4) Remove those situations.
- 5) Execute the modified prediction system.

On evaluation, we found that for the data sets studied here, TEAK generated significantly *better* estimates than comparable methods.

More generally, the success of the easy path principle recommends it for future research. When designing a predictor, it is useful to *first* try optimizing for the situations where prediction is easy, *before* struggling with arcane and complex mechanisms to handle the harder situations. For example, in future work, we will apply steps 1,2,3,4,5 to other aspects of effort estimation like feature weighting, and similarity measures.

The rest of this paper is structured as follows. After a review of the general field of effort estimation, we will focus on ABE (analogy-based estimation). For ABE, we will work through the above five steps to design TEAK. TEAK's performance will then be compared against six other ABE systems. Our conclusion will be to recommend TEAK for effort estimation.

The paper uses the notation of Figure 1.

2 BACKGROUND

2.1 Scope

This paper is *not* a detailed comparison of analogy-based estimation to other estimation methods (for such a large scale comparison of many different methods, the reader is referred to [9], [14]). While we compare our proposed

| Symbol | Explanation |
|--|--|
| ABE | Analogy Based Estimation. |
| ABE0 | A baseline ABE method. |
| NNet | A neural net prediction system with one hidden layer. |
| LR | Linear regression. |
| GAC | Greedy Agglomerative Clustering. |
| TEAK | Test Essential Assumption Knowledge. |
| GAC1, GAC2 | First and second GAC trees within TEAK. |
| x, y | Depending on the context, x and y can refer to two instances/projects in a dataset or alternatively to two vertices in a GAC tree. |
| x_i, y_i | i^{th} features of projects x and y respectively. |
| w_i | Feature weight for the difference of features x_i and y_i in Euclidean distance function. |
| L_x, L_y, L_z | Leaves of the sub-trees whose roots are x, y and z respectively. |
| k_x, k_y, k_z | The number of leaves in L_x, L_y, L_z respectively. |
| k -NN | k Nearest Neighbors. |
| k | An italic k alone refers to analogies, i.e. selected similar projects. |
| b_i, b_j, c_i, c_j | All these symbols are related to discretization of continuous columns. b_i and b_j refer to breakpoints i and j , which in return produce discrete bins that have counts of c_i and c_j instances within themselves. |
| Best(K) | A procedure that heuristically finds the best k value for a dataset. |
| σ_x^2, σ_y^2 | Assuming that x, y and z are vertices in a GAC tree and x is the parent of y and z , σ_x^2 refers to the variance of instances in x and σ_y^2 refers to the weighted sum of the variances of y and z . |
| $\alpha, \beta, \gamma, R, \max(\sigma^2)$ | These symbols are associated with different pruning policies. α, β and γ keep user-defined values to fine-tune pruning. R is a random variable that can have values from 0 to 1. $\max(\sigma^2)$ refers to maximum variance of all sub-trees in a GAC tree. |
| T, N | T refers to a given dataset and N refers to a test set out of this dataset. |
| $predicted_i$ | The effort of test instance $N_i \in N$ predicted by some induced prediction system. |
| $actual_i$ | The actual effort seen in test instance N_i in N |
| AR | Absolute residual. $ actual_i - predicted_i $ |
| MRE | magnitude of relative error. $\frac{AR}{actual_i}$ |
| PRED(X) | The percentage of estimates that are within X% of the actual value. |
| $win_i, tie_i, loss_i$ | The total number of wins, ties and losses of a variant in comparison to other variants according to Wilcoxon signed rank test. |

Fig. 1: The explanations of symbols that are used in our research are summarized here. Symbols that are related to each other are grouped together.

new technique to a limited number of other estimation methods (specifically, neural networks and regression), that comparison is only to ensure that analogy-based estimation is not noticeably worse than other methods in widespread use.

The main point of this paper is as follows. Analogy-based effort estimation is a widely-used and widely-studied technique [8], [12], [15]–[25]. This paper reports a novel method to improve that technique. Specifically, when estimating via analogy, it is *best to first prune all subsets of the data with high variance*. We will argue that a new *variance heuristic* is a better way to select analogies:

- Without this heuristic, analogies are selected by their distance to the test instance.
- With this heuristic, a pre-processor prunes the space of possible analogies, removing the subsets of the data with high variance (in high variance subsets, training data offers highly variable conclusions).

This heuristic works, we believe, since if a test instance falls into such subsets then (by definition) minor changes to the test will lead to large changes in the prediction (due to the variability in that region). We show below that, by removing those problematic subsets, effort estimation by analogy can be improved.

2.2 Analogous Approaches

While the variance heuristic is novel and unique in the effort estimation literature, analogous proposals can be found in the requirements engineering literature, dating back to the 1990s. In the seminal paper “To Be *and* Not To Be”, Nuseibeh [26] discusses a spectrum of methods for handling inconsistent specifications. One method is *circumvent*; i.e. instead of expending effort resolving regions of contradiction, add “pollution markers” that screen the problematic regions away from the rest of the system. Note that our variance heuristic (that prunes the data subsets with high variance) is something like a pollution marker since it guides the reasoning away from problematic training data.

Another use for such marks is to mark any segments that human agents need to explore. Turning back from requirements engineering (which was the focus of Nuseibeh’s discussion) back to effort estimation (which is the focus of this paper), we could utilize pollution marks to highlight regions where more data collection might be beneficial. Note that this approach is not explored here since our premise is that we must make the best use possible of fixed data. However, this might be a promising area of future research.

2.3 Effort Estimation

Having set the context for this paper, we now turn to the details.

After Shepperd [6], we say that software project effort estimation usually uses one of three methods:

- Human-centric techniques (a.k.a. expert judgment);
- Model-based techniques including:
 - Algorithmic/parametric models such COCOMO [1], [27];
 - Induced prediction systems.

Human centric techniques are the most widely-used estimation method [28], 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 [29] reports that humans are surprisingly poor at reflecting and improving on their expert judgments.

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 [9]. 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 [1] 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 linear regression, neural nets, and analogy, just to name a few [9], [30]. Analogy-based estimation is discussed in detail in the next section. In order to give the reader some context, we offer here some notes on none-analogy methods (estimation methods other than analogy based estimation). Regression assumes that the data fits some function. The parameters of that function are then adjusted to minimize the difference between the values predicted by the model and the actual values in the training data. For example, in linear regression, the model is assumed to be of the form:

$$y = \beta_0 + \beta_1x_1 + \beta_2x_2 + \dots$$

where x_i are model inputs, y are the model outputs and β_i are the coefficients adjusted by a linear regression induction system.

Neural nets are useful when the data distributions are not simple linear functions [31]–[33]. An input layer of project details is connected to zero or more “hidden” layers which, in turn, connect to an output node (the effort prediction). The connections are weighted directed edges. If the signal arriving to a node sums to greater than some threshold value, the node is said to “fire” and a weight is propagated across the network. Learning in a neural net compares the output value to the expected value, then applies some correction method to improve the edge weights (e.g. the “back propagation” algorithm first invented by Bryson and Ho in 1969 [34], and made popular by Rumelhart et al. in the 1980s [35]).

All induction systems require a bias in order to decide what details can be safely ignored. For example, linear regression assumes that the effort data fits a straight line. When data does not match the bias of the induction system, various patches have been proposed. Boehm [1, p526-529] and Kitchenham & Mendes [36] advocate taking the logarithms of exponential distributions before applying linear regression. Selecting the right patch is typically a manual process requiring an analyst experienced in effort estimation.

2.4 Analogy-based Estimation (ABE)

In ABE, effort estimates are generated for a *test* project by finding similar completed software projects (a.k.a. the *training projects*). Following Kadoda & Shepperd [19], Mendes et al. [11], and Li et al. [12] we define a baseline ABE called ABE0, as follows.

ABE0 executes over a table of data where:

- Each row contains one project;

- Columns contain *independent* variables (features) in the projects and *dependent* variables (features) that stores, for example, effort and duration required to complete one project.

After processing the training projects, ABE0 inputs one test project then outputs an estimate for that project. To generate that estimate, a *scaling measure* is used to ensure all independent features have the same degree of influence on the distance measure between test and training projects. Also, a *feature weighting* scheme is applied to remove the influence of the less informative independent features. For example, in feature subset selection [37], some features are multiplied by zero to remove redundant or noisy features.

The similarity between the target project case and each case in the case-based repository is determined by a similarity measure. There are different methods of measuring similarity have been proposed for different measurement contexts. A similarity measure is measuring the closeness or the distance between two data objects in an n -dimensional feature space, the result is usually presented in a distance matrix (or similarity matrix) identifying the similarity among all cases in the dataset. The Euclidean distance metric is the most commonly used in ABE for its distance measures, and it is suitable for continuous values such as software size, effort and duration of a project. It is based on the principle of Pythagorean Theorem to derive a straight line distance between two points in n -dimensional space.

In general, the unweighted Euclidean distance between two points $P = (p_1, p_2, \dots, p_n)$ and $Q = (q_1, q_2, \dots, q_n)$, and can be defined and calculated as:

$$\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^n (p_i - q_i)^2} \quad (1)$$

An alternative is to apply different weights to each individual project feature to reflect the its relative importance in the prediction system. The weighted Euclidean distance can be calculated as:

$$\sqrt{w_1(p_1 - q_1)^2 + w_2(p_2 - q_2)^2 + \dots + w_n(p_n - q_n)^2} = \sqrt{\sum_{i=1}^n w_i (p_i - q_i)^2} \quad (2)$$

where w_1 and w_n are the weights of 1^{st} and n^{th} project features. Note that in the special case of $w_i = 1$ (i.e. equal weighting) then the equations are identical.

The above Euclidean distance functions are suitable for general problems, particularly when values are of continuous nature. There are other different distance metrics for non-continuous variable, these include, but are not limited to Jaccard distance for binary distance [38] and Gower distance described by Gower & Legendre [39]. In this paper, we only consider the Euclidean distance measure which is most relevant to the context of software cost estimation.

Irrespective of the similarity measure used, the objective is to rank similar cases from the dataset 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 in the ABE research community [19] [17]. Shepperd & Schofield [17] suggested the ideal value for k is 3, that is, only three closest neighboring cases will be considered. These k cases will be adjusted or adapted to better fit the target problem by predefined rules, a human expert or more commonly, using a simple mean or median of the selected k cases.

2.5 Alternatives to ABE0

Within the space of ABE methods, ABE0 is just one approach. Based on our reading of the literature we see other variants that take different approaches to:

- The selection of relevant features;
- The similarity function;
- The weighting method used in similarity function;
- The case subset selection method (a.k.a selected analogies or k value);
- And the adaption strategy (a.k.a solution function)

Not every paper explores every option. For example:

- In [19], the focus of Kadoda et al. is the impact of the selected number of analogies;
- In [12] Li et al. study the effects of relevant subset selection in training set (i.e. historical data) as well as feature weighting in the similarity function;
- Auer et al. propose an optimal weight finding mechanism by means of extensive search in [8];
- In [15], Walkerden et al. investigate selected analogies and compare the performance of human experts to that of tools;
- Finally in [11], Mendes et al. limit historical data to a single domain and compare different ABE configurations to non-ABE methods.

Generalizing from the above, the following notes try to map the space of options within current research [8], [11], [12], [15], [19]. Since researchers are developing new technologies for effort estimation all the time such as AQUA [40], AQUA+ [20] and COSECKMO [9], this map is incomplete. However, it does illustrate our general point that there are thousands of possible variants to ABE.

2.5.1 Three Case Subset Selectors

A *case subset selection* is sometimes applied to improve the set of training projects. These selection mechanisms are characterized by how many cases they remove:

- *Remove nothing*: Usually, effort estimators use all training projects [17]. ABE0 is using this variant.
- *Outlier* methods prune training projects with (say) suspiciously large values [24]. Typically, this removes a small percentage of the training data.
- *Prototype* methods find, or generate, a set of representative examples that replace the training cases. Typically, prototype generation removes most of the training data. For example, Chang's prototype

generators [41] explored three data sets A, B, C of size 514, 150, 66 instances, respectively. He converted these into new data sets A', B', C' containing 34, 14, 6 prototypes, respectively. Note that the new data sets were very small, containing only 7%, 9%, 9% of the original data.

2.5.2 Eight Feature Weighting Methods

In other work Li et al. [12], and Hall & Holmes [37] review eight different feature weighting schemes. Li et al. uses a genetic algorithm to learn useful feature weights. Hall & Holmes review a variety of methods ranging from WRAPPER (a $O(2^F)$ search through all subsets of F features) to various filters methods (that run much faster than WRAPPER) including their preferred correlation-based method.

In our own work, we have developed yet another feature weighted scheme. The fundamental assumption underlying ABE0 is that projects that are similar with respect to project features will be also similar with respect to project effort. To formally evaluate this hypothesis, Keung et al. [24] developed a more comprehensive solution towards ABE0, called Analogy-X (a.k.a AX). For example, given two distance matrices constructed from the selected predictor variables and the response variable, we can correlate the two matrices and show their distance correlation function. However different ordering of the matrix elements may result different matrix correlations, AX applies Mantel's technique that randomly permute the distance matrix elements 1,000 times to produce randomization statistic distribution. Based on the Mantel correlation, AX selects the project features that improves overall Mantel correlation and uses a set of procedures similar to that of stepwise regression to select the project features that are statistically relevant to the solution space, effectively removes the need for brute force feature selection in the classical ABE0 proposed in [17]. More importantly AX provides a statistical justification as to whether ABE should be used for the dataset under investigation. Keung et al. [24]'s study also concludes that dataset quality and variance within the dataset are influential factors, removing data points with large variance will improve prediction performance.

2.5.3 Five Discretization Methods

Some feature weighting schemes require an initial *discretization* of continuous columns. Discretization divides a continuous range at break points b_1, b_2, \dots , each containing a count c_1, c_2, \dots of numbers [42]. There are many discretization policies in the literature including:

- Equal-frequency, where $c_i = c_j$;
- Equal-width, where $b_{i+1} - b_i$ is a constant;
- Entropy [43];
- PKID [44];
- Do nothing at all.

2.5.4 Six Similarity Measures

Mendes et al. [11] discuss three similarity measures including the weighted Euclidean measure described above, an unweighted variant (where $w_i = 1$) and a "maximum distance" measure that focuses on the single feature that maximizes inter-project distance. Frank et al. [45] offer a fourth similarity measure that uses a triangular distribution that sets to the weight to zero, after the distance is more than " k " neighbors away from the test instance. A fifth and sixth similarity measures are the Minkowski distance measure used in [46] and the mean value of the ranking of each project feature used in [15].

2.5.5 Four Adaptation Mechanisms

With regards to adaptation, the literature reports many approaches including:

- Report the median effort value of the analogies;
- Report the mean dependent value;
- Summarize the adaptations via a second learner; e.g. regression [10], model trees [9], [47] or neural network [48].
- Report a weighted mean where the nearer analogies are weighted higher than those further away [11];

2.5.6 Six Ways to Select Analogies

Li et al. [12] comment that there is much discussion in the literature regarding the number of analogies to be used for estimation. Numerous methods are proposed, which we divide into *fixed* and *dynamic*.

Fixed methods use the same number of analogies for all items in the test set. For example, Li et al. [12] report that a standard fixed method is to always use $1 \leq k \leq 5$ nearest projects:

- $k = 1$ is used by Lipowezky et al. [49] and Walkerdon & Jeffery [15];
- $k = 2$ is used by Kirsopp & Shepperd [50]
- $k = 1, 2, 3$ is used by Mendes et al. [11]

Dynamic methods adjust the number of analogies, according to the task at hand. For example, following advice from Martin Shepperd², Baker [10] tuned k to a particular training set using the following "Best(K)" procedure:

- 1) Select $N \subseteq T$ training projects at random;
- 2) For each $k \in 1..T-N$, compute estimates for $n \in N$;
- 3) Find the k value with least error in step 2.
- 4) When estimating, use the k -nearest neighbors, where k is set by step 3.

As shown in Figure 2, Best(K) recommends k values that are very different to those seen in the standard fixed methods. These results come from three commonly used data sets (Desharnais, NASA93, and the original COCOMO data set from [1]: for notes on these data sets, see the document available at <http://bit.ly/feimYA>).

2. Personal communication.

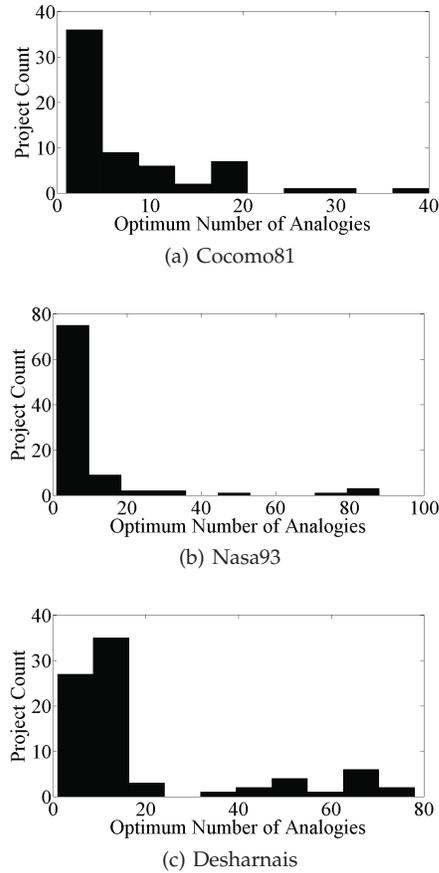


Fig. 2: Distribution of k after removing each project instance, then applying Best(K) on the remaining data. The y-axis counts the number of times a particular k value was found by Best(K).

While ABE systems differ on many aspects, they all use analogy selection. The Figure 2 results suggest that there may be something sub-optimal about standard, widely-used, fixed selection methods. Hence, the rest of this paper takes a closer look at this aspect of ABE.

3 DESIGNING TEAK

The above sample of the literature describes

$$3 \times 8 \times 5 \times 6 \times 4 \times 6 > 17,000$$

ways to implement similarity, adaptation, weighting, etc. Some of these ways can be ruled out, straight away. For example, at $k = 1$, then all the adaptation mechanisms return the same result. Also, not all the feature weighting techniques require discretization, decreasing the space of options by a factor of five. However, even after discarding some combinations, there are still thousands of possibilities to explore. How might we explore all these variations?

The rest of this paper applies the easy path to design and evaluate an ABE system called TEAK (Test Essential Assumption Knowledge). TEAK is an ABE0, with the variations described below.

3.1 Select a Prediction System

Firstly, we *select a prediction system*. We use ABE since:

- It is a widely studied [8], [12], [15]–[25].
- It works even if the domain data is sparse [51].
- Unlike other predictors, it makes no assumptions about data distributions or an underlying model.
- When the local data does not support standard algorithmic/parametric models like COCOMO, ABE can still be applied.

The easy path limits the space of design options to just *those that directly address the essential assumptions of the predictor*. As shown below, for ABE, this directs us to issues of case subset selection and the number of analogies used for estimation.

3.2 Identify Essential Assumption(s)

The second step is to *identify the essential assumptions of that prediction system*. Although it is usually unstated, the basic hypothesis underlying the use of analogy-based estimation is that projects that are similar with respect to project and product factors will be similar with respect to project effort [25]. On the other hand, projects from a high variance region are likely to have very different project effort values and can decrease the accuracy in estimation, which we quote as to “*confuse estimation*” in our research. In other words:

Assumption One: Locality implies homogeneity (for $k > 1$).

This assumption holds for project training data with the following property:

- The k -nearest training projects with effort values E_1, E_2, \dots, E_k have a mean value $\mu = (\sum_i^k E_i) / k$ and a variance $\sigma^2 = (\sum_i^k (E_i - \mu)^2) / (k - 1)$.
- By *Assumption One*, decreasing k also decreases σ^2 .
- If all estimates always have the same distance to the mean μ , then *Assumption One* always fails since in $(\sum_i^k (E_i - \mu)^2) / (k - 1)$, if the numerator is constant then decreasing k will always *increase* the variance.

The core of ABE is the premise that in the neighborhood of training instances, the reductions seen in $(\sum_i^k (E_i - \mu)^2)$ dominates over the increases due to $1/(k - 1)$. As we shall see, this sometimes holds (and sometimes it does not). For example, let us assume that a node x has a left child y and a right child z . Let us further assume that each child has 2 leaves/instances that contain the effort values of $leaves(y) \in \{1253, 1440\}$ staff hours and $leaves(z) \in \{1562, 5727\}$ staff hours, which means that x contains effort values of $\{1253, 1440, 1562, 5727\}$. With these values the variance (σ^2) of each node would be as follows: $\sigma_x^2 = 4.6523e6$, $\sigma_y^2 = 1.7485e4$ and $\sigma_z^2 = 8.6736e6$. In that scenario, going from parent node x to child nodes will create 2 cases:

- 1) Going to y : Reduction seen in $(\sum_i^k (E_i - \mu)^2)$ dominates over the increase due to $1/(k - 1)$, i.e. the

variance decreases when the number of instances decreases.

- 2) Going to z : Reduction seen in $\left(\sum_i^k (E_i - \mu)^2\right)$ cannot dominate over the increase due to $1/(k-1)$, i.e. the variance increases when the number of instances decreases.

As can be seen in this example, it is not necessarily true that moving to a smaller set of neighbors decreases variance. As shown below, it can improve prediction accuracy if ABE takes this matter into account.

3.3 Identify Assumption Violation

The third step is to *recognize situations that violate the essential assumption*. Implementing this step requires some way to compare the variance of larger- k estimates to smaller- k estimates. One way to achieve this is to use some clustering methods that generates a tree of clusters, where each sub-tree contains training data that is closer together than the super-tree.

There are many algorithms for generating trees of clusters. The basic method, called greedy agglomerative clustering (GAC) is used in various fields (data mining [52], databases [53] bioinformatics [54]). GAC executes bottom-up by grouping together at a higher level ($i+1$) the closest pairs found at level i . The algorithm terminates when some level i is found with only one node. GAC is “greedy” in that it does not pause to consider optimal pairings for vertices with very similar distances, i.e. it never backtracks looking for (say) better pairings at level i to reduce the distance between nodes at level $i+1$.

The result of GAC is a tree like Figure 3. Note that, in this tree, the original training data are found at the leaves of the tree. All other nodes are nodes artificially generated by GAC to represent the median of pairs of the leaves, the median of the medians, and so on (recursively).

A GAC tree can be viewed as a tree of clusters where each node at height i is the centroid of the sub-clusters at height $i-1$. Given T initial instances, GAC builds a trees of maximum height $\log_2(T)$. Since the number of vertices is halved at each next level, building a GAC tree requires the following number of distance calculations:

$$\left(\sum_i^{\log_2(T)} \left(\frac{T}{2^{i-1}}\right)^2\right) = \frac{4}{3}(T^2 - 1)$$

This $O(T^2)$ computation is deprecated for large T . However, for this study, GAC construction takes less than a second³. Our runtimes were fast because effort estimation data sets are usually very small: Figure 4 shows all our training projects contain less than 100 examples.

There are many other, faster, algorithms for generating trees of clusters including bisecting k-means [55] which,

3. Using Matlab on a standard Intel x86 dual core notebook running LINUX with 4GB of ram.

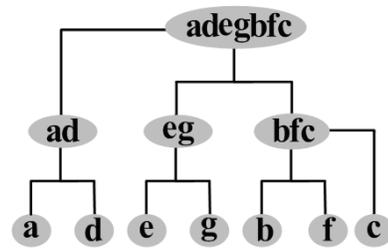


Fig. 3: A sample GAC tree built from 7 instances: a , b , c , d , e , f and g . In each level the closest two nodes are coupled-up to produce the nodes in one higher level. However, odd numbered instances result in unbalanced trees. In this figure, when all leaf nodes are paired up (pairing of letters here is random and just for illustration), c remains alone. The alone remaining node at any level, will find the closest node to itself in one higher level and will merge to that node. For example c merges to the node containing b and f .

at each level, calls k-means⁴ with $k=2$ several times to split one cluster into two. The division leading to the clusters with the best intra-cluster similarities are then stored and used in subsequent splits. Other approaches, like MESO [56], recursively divide the data into smaller and smaller spheres containing *close* instances. MESO uses an incremental method to learn and update what *close* means for a particular data set.

We use GAC (with the distance measure of 1) rather than other methods like bisecting k-means or MESO for two reasons. Firstly, these other methods use various heuristics to improve their runtimes. Our data sets are so small that such heuristic methods are not necessary. Secondly, our results (described below) with GAC are so promising that we are not motivated to experiment beyond GAC.

Using a GAC tree, finding the k -nearest neighbors in project data can be implemented using the following TRAVERSE procedure:

- 1) Place the test project at the root of the GAC tree.
- 2) Move the test project to the nearest child (where “nearest” is defined by Equation 1).
- 3) Go to step 2

Clearly, a $k=1$ nearest-neighbor estimate comes from TRAVERSE-ing to a leaf, then reporting the effort of that leaf. More generally, a $k=N$ nearest-neighbor estimate comes from TRAVERSE-ing to a sub-tree with N leaves, then reporting the median efforts of those leaves.

TRAVERSE can test *Assumption One*. Let some current vertex x have children y and z . We say that:

- The sub-trees starting at x, y, z have leaves L_x, L_y, L_z (and $L_x = L_y \cup L_z$).
- The number of sub-tree leaves is $k_x = k_y + k_z$.

4. The k-means clustering algorithm selects centroids at random, labels each instance by its nearest centroid, then updates the centroid position to the central position of all instances with the same label. The algorithm repeats till the centroid position stabilizes.

- The variance of the leaves' efforts are $\sigma_x^2, \sigma_y^2, \sigma_z^2$.
- After C4.5 [57], we say the variance of the trees below x (denoted σ_{yz}^2) is the weighted sum:

$$\sigma_{yz}^2 = \frac{k_y}{k_x} \sigma_y^2 + \frac{k_z}{k_x} \sigma_z^2$$

Parent trees have the nodes of their children (plus one). If we TRAVERSE from a parent x to a child, then the sub-tree size k decreases. That is, TRAVERSE-ing moves into progressively smaller sub-trees.

Assumption One holds if, when TRAVERSE-ing from all vertices x with children y and z , the sub-tree variance decreases. That is:

$$\forall x \in T : \sigma_x^2 > \sigma_{yz}^2 \quad (3)$$

Note one special case of the above: In the case of $k = 1$, variance is zero since, by definition, all members of a sample of size one are the same. Hence, under that scenario TEAK would mostly return leaf nodes, which would lead to an erroneous execution. This scenario was realized before the implementation phase and it was addressed with a control mechanism. The control mechanism lets the test instance go down the GAC tree at most until one level higher than the leaves. Therefore, the variance value that plays a decisive role on the movement of test instances always comes from a population of at least two instances.

3.4 Remove Violations

The fourth step in TEAK's design is to *remove the situations that violate the essential assumption*. We instrumented TRAVERSE to report examples where Equation 3 was violated; i.e. where it recursed into sub-trees with a larger variance than the parent tree. We found that this usually occurs if a super-tree contains mostly similar effort values, but one sub-tree has a minority of outliers. For example:

- Suppose some vertex x has children y, z .
- Let each child start sub-trees whose leaves contain the effort values $leaves(y) \in \{1253, 1440\}$ staff hours and $leaves(z) \in \{1562, 5727\}$ staff hours.

In this example:

- The leaves of the parent tree x have similar effort values: 1,253 and 1,562 and 1,440 staff hours.
- But the leaves of the subtree z has outlier values; i.e. 5,727.
- TRAVERSE-ing from the super-tree x to the sub-tree z increases the variance by two orders of magnitude.

A *sub-tree pruning policy* is used to prune sub-trees with a variance that violates the essential assumption. We experimented with various policies that removed subtrees if they had:

- 1) more than α times the parent variance;
- 2) more than $\beta * \max(\sigma^2)$;
- 3) more than $R^\gamma * \max(\sigma^2)$, where R is a random number $0 \leq R \leq 1$.

In order to avoid over-fitting, our pruning policy experiments were restricted to one data set (Boehm's CO-COMO embedded projects [1]) then applied, without modification, to the others. The randomized policy (#3) produced lowest errors, with smallest variance. The success of this randomized policy suggests two properties of effort estimation training data:

- The boundary of "good" training projects is not precise. Hence, it is useful to sometimes permit random selection of projects either side of the boundary.
- The policy tuning experiments recommended $\gamma = 9$. This selects for subtrees with less than 10% of the maximum variance⁵. This, in turn, suggests that the above example is typical of effort estimates; i.e. sub-tree outliers are usually a few large effort values.

In theory, stochastic methods like policy #3 introduce a degree of instability in the performance of the induction system. In practice, this is not an issue with TEAK. In the evaluation section, described below, we repeat out analysis of TEAK 20 times using various performance measures and experimental rigs. When we compare the results of those repeated trials against just running TEAK once, we can see no major performance differences.

3.5 Execute the Modified System

The final step in the design of TEAK is to build a new prediction system. TEAK executes as follows:

- Apply GAC to the training projects to build a tree called GAC1;
- Prune GAC1 using the sub-tree pruning policy described above. The remaining leaves are the *prototypes* to be used in effort estimation.
- Apply GAC to the prototypes to build a second tree called GAC2.
- Place the test project at the root of GAC2. Compute an estimate from the median value of the GAC2 projects found by TRAVERSE2. TRAVERSE2 is a variant of TRAVERSE that ensures the essential assumption is never violated. It stops recursing into GAC2 sub-trees when Equation 3 is violated.

4 COMPARISONS

Recall the pre-experimental concern expressed above: Taking the easy path might ignore important design issues, to the detriment of the predictions. To address that concern, this section compares TEAK to a range of other ABE0 variants as well the other induced prediction systems described in §2; i.e. neural nets and linear regression (we selected this particular range of algorithms at the suggestion of reviewers of this paper).

As discussed above in §2.1, a detailed comparison of analogy-based estimation to other estimation methods is not the fundamental aim of this research. To observe the results of such a detailed comparison, we found [9]

5. The mean of $rand()^9 \approx 0.1$.

| Dataset | Features | $T = Projects $ | Content | Historical Effort Data | | | | | |
|---------------|----------|------------------|-------------------------------|------------------------|-----|--------|------|-------|----------|
| | | | | Units | Min | Median | Mean | Max | Skewness |
| Cocomo81 | 17 | 63 | NASA projects | months | 6 | 98 | 683 | 11400 | 4.4 |
| Cocomo81e | 17 | 28 | Cocomo81 embedded projects | months | 9 | 354 | 1153 | 11400 | 3.4 |
| Cocomo81o | 17 | 24 | Cocomo81 organic projects | months | 6 | 46 | 60 | 240 | 1.7 |
| Nasa93 | 17 | 93 | NASA projects | months | 8 | 252 | 624 | 8211 | 4.2 |
| Nasa93c2 | 17 | 37 | Nasa93 projects from center 2 | months | 8 | 82 | 223 | 1350 | 2.4 |
| Nasa93c5 | 17 | 40 | Nasa93 projects from center 5 | months | 72 | 571 | 1011 | 8211 | 3.4 |
| Desharnais | 12 | 81 | Canadian software projects | hours | 546 | 3647 | 5046 | 23940 | 2.0 |
| SDR | 22 | 24 | Turkish software projects | months | 2 | 12 | 32 | 342 | 3.9 |
| Albrecht | 7 | 24 | Projects from IBM | months | 1 | 12 | 22 | 105 | 2.2 |
| ISBSG-Banking | 14 | 29 | Banking projects of ISBSG | hours | 662 | 2355 | 5357 | 36046 | 2.6 |
| Total: 448 | | | | | | | | | |

Fig. 4: The 448 projects used in this study come from 10 data sets. Indentation in column one denotes a dataset that is a subset of another dataset. For notes on this data, see the document available at <http://bit.ly/feimyA>.

and [14] particularly helpful. In this research we compare TEAK to other estimation methods (specifically, neural networks and regression), merely to ensure that analogy-based estimation is not significantly *worse* than other methods in widespread use.

In the following comparisons, TEAK will be assessed using:

- Two different experimental rigs (leave-one-out and n-way cross-validation: see §4.1);
- Three different performance measures (AR, MRE, PRED(25): see §4.2);

4.1 Randomized Trials

Recall that TEAK uses a randomized method for subtree pruning. Any evaluation of such a randomized method must be repeated multiple times. Appealing to the central limit theorem, we used twenty repeats.

- Twenty times, for each data set, we randomize the order of the rows in that data set.
- Next, we conducted *both* a Leave-One-Out study and a 3-way cross-validation study.

In Leave-One-Out, given T projects, then $\forall t \in T$, use t as the test project and the remaining $T - 1$ projects for training. In 3-way cross-validation, the data set of T projects is divided into three bins, bin_i is used for testing while the remaining $T - bin_i$ projects are used for testing.

Since some of our data sets are very small (e.g. the 24 instances of Cocomo81o), we used a 3-way cross validation (and not the 10-way used by, say, Quinlan [57]). Some thought was given to using 3-way for small data sets and 10-way for larger data sets. However, this would introduce a complication into the analysis that is neither recommended by the literature, nor handled by any statistical technique that we are aware of.

We use both Leave-one-out and N-way cross-validation since the effort estimation literature is ambiguous on which is most appropriate. In the Kitchenham et al. survey [7], all the projects reviewed in their Table 3 used N-way cross-validation. However, other prominent studies prefer leave-one-out [22].

For these studies, we used all independent features when computing similarities. We applied twenty randomized trials using the 448 projects from 10 data sets of

Figure 4 (for notes on this data, see the document available at <http://bit.ly/feimyA>). In all, these randomized trials generated a total of 8,990 training/test set pairs, where $20 \times 448 = 8,960$ of the pairs come from Leave-One-Out and another $10 \times 3 = 30$ pairs come from 10*3-way.

4.2 Details

For each of these 8,990 pairs of training/test, estimates were generated by TEAK, neural networks, regression, and six other ABE0 variants:

- Five variants returned the median effort seen in the k -th nearest neighbors for $k \in \{1, 2, 4, 8, 16\}$.
- The other variant returned the median effort seen in k neighbors found using Baker’s Best(K) procedure. From §2.5, recall that Best(K) adjusts k to each data set by reflecting over all the training projects.
- For notes on regression and NNet please see section 2.3.

Since this is paired data (same train and test data passed through multiple treatments), we applied a Wilcoxon signed rank test (95% confidence) to rank the resulting estimates. Ranked⁶ statistical tests like the Wilcoxon are useful if it is not clear that the underlying distributions are Gaussian [58]. Because, ranked statistics mitigate the problem of effort estimation results that sometimes contain a small number of very large errors [59].

We collected information on three performance metrics: AR, MRE, PRED(25). The magnitude of the absolute residual (AR) is computed from the difference between predicted and actual:

$$AR = |actual_i - predicted_i| \quad (4)$$

We prefer AR to other performance measures since we share Shepperd’s concern [60] that anything other than the simplest evaluation statistic can introduce analysis issues. Nevertheless, other measures are more common in the effort estimation literature (e.g. see Table 3 of [7]) such as MRE and PRED(25). MRE is the magnitude of the relative error. It is calculated by expressing AR as a ratio of the actual prediction:

$$MRE = \frac{|actual_i - predicted_i|}{actual_i} \quad (5)$$

6. In a ranking analysis, the raw results (10.2,21.3,22.1,24.25,30,100) are replaced with their ranks in the sort order; i.e. (1,2,3,4,5,6,7).

```

wini = 0, tiei = 0, lossi = 0
winj = 0, tiej = 0, lossj = 0
if WILCOXON(Pi, Pj) says they are the same then
    tiei = tiei + 1;
    tiej = tiej + 1;
else
    if median(Pi) < median(Pj) then
        wini = wini + 1
        lossj = lossj + 1
    else
        winj = winj + 1
        lossi = lossi + 1
    end if
end if

```

Fig. 5: Pseudocode for win-tie-loss calculation between variant i and j with performance measures P_i and P_j . Note here that only for Pred(25) the comparison is based on actual values ($Pred(25)_i$, $Pred(25)_j$) rather than median values ($median(P_i)$, $median(P_j)$).

AR and MRE are calculated for every item in a test set. PRED(X), on the other hand, is a summary statistic that reports behavior over an entire test suite. PRED(X) reports the average percentage of the N estimates in the test set that were within X% of the actual values: For example, PRED(30)=50% means that half the estimates are within 30% of the actual value. Chulani & Boehm assesses his models using PRED(30) [61]. We use the stricter criteria of PRED(25) since that is more common in the literature; e.g. [17], [33], [62].

In order to summarize the results of the Wilcoxon comparisons of the MRE, AR, PRED(25) measures, we use the following win-tie-loss procedure. For each iteration of the randomized trials, each data set generated $20 \times 7 = 140$ (MRE, PRED(25), AR) distributions for each induced prediction system (neural nets, regression, the ABE0 variants). To calculate the win-tie-loss values, we first checked if two distributions i, j are statistically different according to the Wilcoxon test. If not, then we incremented tie_i and tie_j . On the other hand, if they turned out to be different, we updated win_i, win_j and $loss_i, loss_j$ after a numerical comparison of their median values. The pseudocode for win-tie-loss calculation is given in Figure 5.

In median and mean performance measures used in other studies (see Table 3 of [7]), the entire distribution is summarized by its central tendency (measured in terms of median or mean), then two methods are compared solely in terms of those two central points. On the approach of Figure 5 on the other hand, with the use of Wilcoxon the variance around centrality is also considered, which is not the case with single-point assessment methods. Such single point assessments are also deprecated in literature; e.g. see Foss et al.'s scathing critique of mean MRE [63]. Therefore, we adopt the approach of Figure 5 over single-point assessments.

| Data set | Variant | Win | Tie | Loss | Win - Loss |
|---------------|---------|-----|-----|------|------------|
| Cocomo81 | TEAK | 87 | 73 | 0 | 87 |
| | Best(K) | 49 | 110 | 1 | 48 |
| | k=16 | 42 | 107 | 11 | 31 |
| | k=8 | 41 | 100 | 19 | 22 |
| | k=4 | 28 | 96 | 36 | -8 |
| | NNet | 37 | 76 | 47 | -10 |
| | k=1 | 28 | 88 | 44 | -16 |
| | k=2 | 26 | 82 | 52 | -26 |
| Cocomo81e | LR | 7 | 18 | 135 | -128 |
| | TEAK | 55 | 105 | 0 | 55 |
| | NNet | 43 | 117 | 0 | 43 |
| | k=8 | 32 | 126 | 2 | 30 |
| | k=16 | 32 | 126 | 2 | 30 |
| | Best(K) | 32 | 126 | 2 | 30 |
| | k=4 | 18 | 113 | 29 | -11 |
| | k=1 | 8 | 97 | 55 | -47 |
| Cocomo81o | k=2 | 4 | 101 | 55 | -51 |
| | LR | 11 | 59 | 90 | -79 |
| | TEAK | 136 | 0 | 0 | 24 |
| | k=16 | 9 | 151 | 0 | 9 |
| | k=8 | 8 | 152 | 0 | 8 |
| | Best(K) | 8 | 152 | 0 | 8 |
| | NNet | 9 | 150 | 1 | 8 |
| | k=4 | 7 | 151 | 2 | 5 |
| Nasa93c5 | LR | 7 | 145 | 8 | -1 |
| | k=2 | 2 | 128 | 30 | -28 |
| | k=1 | 1 | 125 | 34 | -33 |
| | TEAK | 40 | 120 | 0 | 40 |
| | LR | 25 | 135 | 0 | 25 |
| | k=16 | 17 | 141 | 2 | 15 |
| | Best(K) | 17 | 139 | 4 | 13 |
| | k=8 | 16 | 134 | 10 | 6 |
| SDR | NNet | 10 | 144 | 6 | 4 |
| | k=4 | 10 | 127 | 23 | -13 |
| | k=2 | 7 | 110 | 43 | -36 |
| | k=1 | 3 | 100 | 57 | -54 |
| | TEAK | 67 | 93 | 0 | 67 |
| | k=1 | 43 | 97 | 20 | 23 |
| | NNet | 25 | 123 | 12 | 13 |
| | k=4 | 26 | 118 | 16 | 10 |
| ISBSG-Banking | k=8 | 18 | 132 | 10 | 8 |
| | k=2 | 20 | 126 | 14 | 6 |
| | Best(K) | 16 | 126 | 18 | -2 |
| | k=16 | 13 | 120 | 27 | -14 |
| | LR | 0 | 49 | 111 | -111 |
| | TEAK | 30 | 130 | 0 | 30 |
| | NNet | 24 | 136 | 0 | 24 |
| | LR | 23 | 137 | 0 | 23 |
| Nasa93 | k=16 | 22 | 138 | 0 | 22 |
| | k=8 | 19 | 141 | 0 | 19 |
| | Best(K) | 21 | 137 | 2 | 19 |
| | k=4 | 14 | 112 | 34 | -20 |
| | k=1 | 8 | 106 | 46 | -38 |
| | k=2 | 4 | 73 | 83 | -79 |
| | LR | 72 | 88 | 0 | 72 |
| | TEAK | 26 | 134 | 0 | 26 |
| Nasa93c2 | NNet | 16 | 143 | 1 | 15 |
| | k=16 | 13 | 133 | 14 | -1 |
| | k=8 | 15 | 128 | 17 | -2 |
| | Best(K) | 14 | 128 | 18 | -4 |
| | k=4 | 6 | 122 | 32 | -26 |
| | k=2 | 4 | 113 | 43 | -39 |
| | k=1 | 6 | 107 | 47 | -41 |
| | LR | 158 | 2 | 0 | 158 |
| Desharnais | TEAK | 36 | 106 | 18 | 18 |
| | k=16 | 25 | 115 | 20 | 5 |
| | NNet | 17 | 123 | 20 | -3 |
| | k=8 | 15 | 116 | 29 | -14 |
| | Best(K) | 15 | 116 | 29 | -14 |
| | k=4 | 11 | 101 | 48 | -37 |
| | k=2 | 5 | 95 | 60 | -55 |
| | k=1 | 6 | 90 | 64 | -58 |
| Desharnais | LR | 63 | 97 | 0 | 63 |
| | NNet | 51 | 109 | 0 | 51 |
| | TEAK | 37 | 121 | 2 | 35 |
| | k=16 | 25 | 129 | 6 | 19 |
| | k=8 | 22 | 124 | 14 | 8 |
| | Best(K) | 16 | 120 | 24 | -8 |
| | k=4 | 14 | 116 | 30 | -16 |
| | k=2 | 6 | 80 | 74 | -68 |
| Desharnais | k=1 | 1 | 74 | 85 | -84 |

Fig. 6: MRE based win-loss-tie results from the 20*Leave-One-Out experiments. For each data set results are sorted by win minus loss values. Gray cells indicate variants with zero losses. The performance of the various induced prediction systems is summarized in top-left corner of Figure 7

4.3 Results

Initially, our intention was to report results using all the data sets of Figure 4. However, we found that the AI-

| | 20 * LEAVE-ONE-OUT | | | | | | | | 20 * 3-WAY CROSS-VALIDATION | | | | | | | | | |
|-----------------|--------------------|----|------|---------|-----|------|-----|-----|-----------------------------|------|----|------|---------|-----|------|-----|-----|-----|
| | TEAK | LR | NNet | Best(K) | k=1 | k=16 | k=2 | k=4 | k=8 | TEAK | LR | NNet | Best(K) | k=1 | k=16 | k=2 | k=4 | k=8 |
| MRE | | | | | | | | | | | | | | | | | | |
| Cocomo81 | ▲ | | | | | | | | | ▲ | | | | | | | | |
| Cocomo81e | ▲ | | | | | | | | | ▲ | | | | | | | | |
| Cocomo81o | ▲ | | | | | | | | | ▲ | | | | | | | | |
| Nasa93 | | ▲ | | | | | | | | | ▲ | | | | | | | |
| Nasa93c2 | | ▲ | | | | | | | | | ▲ | | | | | | | |
| Nasa93c5 | ▲ | | | | | | | | | ▲ | | | | | | | | |
| Desharnais | | ▲ | | | | | | | | | ▲ | | | | | | | |
| Sdr | ▲ | | | | | | | | | ▲ | | | | | | | | |
| ISBSG-Banking | ▲ | | | | | | | | | ▲ | | | | | | | | |
| Count | 6 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 6 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Pred(25) | | | | | | | | | | | | | | | | | | |
| Cocomo81 | ▲ | | | | | | | | | ▲ | | | | | | | | |
| Cocomo81e | ▲ | | | ▲ | | | | | | ▲ | | ▲ | | | | | | |
| Cocomo81o | ▲ | | | | | | | | | ▲ | | | | | | | | |
| Nasa93 | | ▲ | | | | | | | | | ▲ | | | | | | | |
| Nasa93c2 | | ▲ | | | | | | | | | ▲ | | | | | | | |
| Nasa93c5 | ▲ | | | | | | | | | ▲ | | | | | | | | |
| Desharnais | | ▲ | | | | | | | | | ▲ | | | | | | | |
| Sdr | ▲ | | | | | | | | | ▲ | | | | | | | | |
| ISBSG-Banking | ▲ | | | | | | | | | ▲ | | | | | | | | |
| Count | 5 | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| AR | | | | | | | | | | | | | | | | | | |
| Cocomo81 | ▲ | | | | | | | | | ▲ | | | | | | | | |
| Cocomo81e | ▲ | | | | | | | | | ▲ | | | ▲ | | | | | |
| Cocomo81o | ▲ | | | | | | | | | ▲ | | | | | | | | |
| Nasa93 | | ▲ | | | | | | | | | ▲ | | | | | | | |
| Nasa93c2 | | ▲ | | | | | | | | | ▲ | | | | | | | |
| Nasa93c5 | ▲ | | | | | | | | | ▲ | | | | | | | | |
| Desharnais | | ▲ | | | | | | | | | ▲ | | | | | | | |
| Sdr | ▲ | | | | | | | | | ▲ | | | | | | | | |
| ISBSG-Banking | ▲ | | | | | | | | | ▲ | | | | | | | | |
| Count | 6 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 6 | 3 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

Fig. 7: Summary of the random trials; e.g. Figure 6 is summarized top-left in “MRE”. This figure displays the top performing inducted predictive system, measured via $(win - loss)$. This is repeated for all the performance measures (MRE, PRED(25), AR) and both experimental rigs (leave-one-out on the left and 3-way cross-validation on the right). The last row of each table shows the sum of times a method appeared as the top performing variant. In the majority of cases, TEAK appears as the top-ranked predictive system.

brecht data set was producing a very large number of ties (over 98%). On closer inspection, we found that in our rig, Albrecht was a data set in which all our treatments generated very similar results (the plots of the MREs generated by our eight methods was indistinguishable). Since Albrecht was mostly unable to distinguish between the different treatments, we excluded it from the rest of our analysis.

The resulting Win/Loss/Ties values from a Leave-One-Out study that measured MRE are shown in Figure 6. When ranked in terms of $win - loss$, in $\frac{6}{9}$ data sets, TEAK is the *top ranked* method; i.e it always ranked first on that performance score. The next best method was linear regression that is found in the top rank in only $\frac{3}{9}$ data sets.

These scores are summarized top left of Figure 7 (see the tables for “MRE”). There is not enough space in this article to repeat Figure 4 for every combination of (Leave-One-Out, Cross-Val)*(MRE, AR, PRED(25)); i.e. six times in all. Hence, we present a summary of those results in Figure 7. In all cases:

- 1) Best(K) and $K \in \{1, 2, 4, 8, 16\}$ rarely appeared in the top ranked methods. That is, standard analogy selection mechanism performed comparatively worse than applying TEAK’s variance heuristic.
- 2) While non-analogy methods sometimes did better on certain data sets, overall, TEAK’s extension to analogy-based reasoning was competitive with

non-analogy methods.

From result 1), we recommend variance pruning for analogy estimation since unequivocally, of all the analogy variants studied here, TEAK is the superior system.

As to result 2), we hesitate to conclude, just from this sample, that TEAK is *always* the best effort estimation method. However, its results are encouraging and should motivate continued research into analogy-based methods. In our review of effort estimation [9], we commented that best practices include generating estimates from multiple sources. Certainly, these results offer no reason to *exclude* analogy as one of those sources.

4.4 But Why Does it Work?

In discussions over TEAK, we are sometimes asked if it is wise to use variance to assess the suitability of neighborhood for providing donor cases. The argument goes as follows: While a high variance for a given neighborhood of k suggests that this is a *bad* neighborhood, a low variance does not necessarily imply that the neighborhood is *good*.

In reply, we note that TEAK does not *only* use variance to select the donor cases. TRAVERSE2 pushes *away* from regions with high variance while pushing *towards* regions with similar features to the test instance. TRAVERSE2 pushes away from high variance regions since:

- It executes over a space of training data which high variance regions pruned away;

- Its recursive descent terminates if it enters a region of increasing variance.

At the same time, TRAVERSE2 pushes towards regions with similar features as follows:

- It descends a binary tree of clusters.
- At each step, the test instance is moved towards the sub-tree whose Euclidean distance is closest to the test instance.

The above experiments show, we argue, that this policy does better than just pushing towards regions with higher similarity. That is, *augmenting* nearest neighbor algorithms with variance avoidance does better than just applying nearest neighbor.

4.5 Threats to Validity

Internal validity questions to what extent the cause-effect relationship between dependent and independent variables hold [64].

The general internal validity issue is that data mining experiments (like those discussed above) do not collect new data, but only generates theories from historical data. Ideally, we should take a learned theory and apply it to some new situation, then observe if the predicted effect occurs in practice. Note that if no explicit theory is generated, then it cannot be applied outside of the learning system. That is, all ABE systems suffer from issues of internal validity since they do not generate an explicit theory. However, it is possible to mitigate this problem by simulating how an ABE system might be applied to a new situation. Note that the Leave-One-Out approach used in this paper generates estimates using test data that is not used in training.

Construct validity (i.e. face validity) assures that we are measuring what we actually intended to measure [65]. In our research we are using a variety of performance measures (AR, MRE, PRED(25) and a pair of evaluation experiments (leave-one-out and cross-val). This was done to increase the construct validity of this study. MRE is widely used for assessing the performance of competing software effort estimation models [63], [66], [67]. Foss et al. [63] have provided an extensive discussion demonstrating that by using only MRE itself may be leading to incorrect evaluation. Hence, we take care to apply multiple performance measures and a pair of evaluation experiments.

External validity is the ability to generalize results outside the specifications of that study [68]. To ensure the generalizability of our results, we studied a large number of projects. Our datasets contain a wide diversity of projects in terms of their sources, their domains and the time period they were developed in. For example, we used datasets composed of software development projects from different organizations around the world to generalize our results [69]. Our reading of the literature is that this study uses more project data, from more sources, than numerous other papers. All the papers we have read, as well as, Table 4 of [7] list the total number

of projects in all data sets used by other studies. The median value of that sample is 186, which is less than half the 448 projects used in our study.

4.6 Future Work

In this paper, we have applied the easy path principle to design a new method for case & analogy selection. In future work, we will apply the easy path to similarity measures, feature weighting, and adaptation. For example:

- After grouping together rows with similar estimates, we might weight features by their variance within each group (and higher variance means lower weight).
- Alternatively, Lipowezky [49] observes that feature and case selection are similar tasks (both remove cells in the hypercube of all cases times all columns). Under this view, it should be possible to convert our case selector to a feature selector.

Our investigations in this area are very preliminary and, at this time, we have no conclusive results to report.

Another promising avenue to explore is variations on the GAC clustering. Since our results have so far been quite promising, we have not explored alternatives to GAC. For example, we form links between quite distinct clusters merely because they have the minimum average-linkage-clustering (mean value of the distance between instances in each cluster). Perhaps another, more sophisticated, clustering algorithm would be a better way to group data.

5 CONCLUSION

In response to the growing number of options for designing software effort estimators, various researchers (e.g. [9], [10], [12]) have proposed elaborate and CPU-intensive search tools for selecting the best set of design options for some local data. While useful, these tools offer no insight into the effort estimation task: They report *what* the design is in simplifying future effort estimation tasks, but not *why* they were useful. Such insights are useful for reducing the complexity of future effort estimations.

In order to avoid the computation cost of these tools, and to find the insights that simplify effort estimation, we design TEAK using an *easy path* principle. The easy path has five steps.

1. *Select a prediction system:* Analogy-based effort estimation, or ABE, is a widely-studied method that works on sparse data sets. Hence, we selected ABE as our prediction system.

2. *Identify the predictor's essential assumption(s):* The essential assumption of ABE is that *locality implies homogeneity* (for $k > 1$); i.e. the closer the test project approaches the training projects, the smaller the variance in that neighborhood.

3. *Recognize when those assumption(s) are violated:* Mathematically, this can be tested by recursively clustering

project data into a tree whose leaves contain historical effort data and whose internal nodes are medians of pairs of child nodes. When descending this tree, the essential ABE assumption is violated when sub-trees have a larger variance than the parents.

4. *Remove those situations:* This assumptions can be removed by pruning sub-trees with the larger variances.

5. *Execute the modified prediction system:* TEAK builds a second tree of clusters using just the projects not found in high variance sub-trees. Estimates are generated from this second tree by a recursive descent algorithm that stops before the sub-tree variance is higher than the super-tree variance. The leaves of terminating sub-tree are then accessed and the estimate is calculated from the median of the effort values in those leaves.

A pre-experimental concern with the easy path was that, in ignoring the hard training cases, we would miss important aspects of the data. Our experiments do not support that concern. TEAK never lost against other ABE methods and always won the most. Also, TEAK performed at least as well (if not better) than certain other non-analogy-based estimation methods.

Our conclusions are two-fold:

- For those using analogy-based estimation, we strongly recommend pruning instances from regions of high variance prior to generating estimates.
- For those designing new data algorithms, we conclude that it may be detrimental to obsess on the hard cases. Rather, it may be better to enhance what a predictor does best. For example, in the case of ABE, case selection via variance significantly improved the estimates.

REFERENCES

- [1] B. W. Boehm, *Software Engineering Economics*. Prentice Hall PTR, 1981.
- [2] C. Kemerer, "An empirical validation of software cost estimation models," *Comm. of the ACM*, vol. 30, pp. 416–429, May 1987.
- [3] Sparerref.com, "Nasa to shut down checkout & launch control system," August 26, 2002, <http://bit.ly/eiYxlf>.
- [4] B. Boehm, C. Abts, and S. Chulani, "Software development cost estimation approaches - a survey," *Annals of Software Engineering*, vol. 10, pp. 177–205, 2000.
- [5] M. Jorgensen and M. Shepperd, "A systematic review of software development cost estimation studies," *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 33–53, 2007.
- [6] M. Shepperd, "Software project economics: a roadmap," in *Future of Software Engineering*, 2007, pp. 304–315.
- [7] B. Kitchenham, E. Mendes, and G. H. Travassos, "Cross versus within-company cost estimation studies: A systematic review," *IEEE Trans. Softw. Eng.*, vol. 33, no. 5, pp. 316–329, 2007.
- [8] 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.
- [9] T. Menzies, Z. Chen, J. Hihn, and K. Lum, "Selecting best practices for effort estimation," *IEEE Trans. Softw. Eng.*, vol. 32, pp. 883–895, 2006.
- [10] D. Baker, "A hybrid approach to expert and model-based effort estimation," Master's thesis, LCSEE, West Virginia University, 2007, available from <http://bit.ly/hWDEfU>.
- [11] E. Mendes, I. D. Watson, C. Triggs, N. Mosley, and S. Counsell, "A comparative study of cost estimation models for web hypermedia applications," *Emp. Softw. Eng.*, vol. 8, no. 2, pp. 163–196, 2003.
- [12] 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.
- [13] J. R. Quinlan, "Boosting first-order learning," in *7th International Workshop on Algorithmic Learning Theory*, ser. LNAI, vol. 1160. Berlin: Springer, 1996, pp. 143–155.
- [14] T. Menzies, O. Jalali, J. Hihn, D. Baker, and K. Lum, "Stable rankings for different effort models," *ASE*, no. 4, December 2010.
- [15] F. Walkerdien and R. Jeffery, "An empirical study of analogy-based software effort estimation," *Empirical Softw. Engg.*, vol. 4, no. 2, pp. 135–158, 1999.
- [16] C. Kirsopp, M. Shepperd, and R. Premraj, "Case and feature subset selection in case-based software project effort prediction," *Intl. Conf. on Knowledge-Based Sys. and Applied A.I.*, 2003.
- [17] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," *IEEE Trans. Softw. Eng.*, vol. 23, no. 11, pp. 736–743, 1997.
- [18] M. Shepperd, C. Schofield, and B. Kitchenham, "Effort estimation using analogy," in *ICSE '96*, 1996.
- [19] G. Kadoda, M. Cartwright, and M. Shepperd, "On configuring a case-based reasoning software project prediction system," *UK CBR Workshop, Cambridge, UK*, pp. 1–10, 2000.
- [20] J. Li and G. Ruhe, "Analysis of attribute weighting heuristics for analogy-based software effort estimation method aqua+," *Empirical Softw. Engg.*, vol. 13, pp. 63–96, February 2008.
- [21] —, "A comparative study of attribute weighting heuristics for effort estimation by analogy," *ISESE'06*, p. 74, 2006. [Online]. Available: <http://bit.ly/e9yCTR>
- [22] —, "Decision support analysis for software effort estimation by analogy," in *PROMISE '07: Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, 2007, p. 6.
- [23] J. W. Keung, "Empirical evaluation of analogy-x for software cost estimation," in *ESEM '08*, 2008, pp. 294–296.
- [24] J. W. 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, pp. 471–484, 2008.
- [25] J. W. Keung and B. Kitchenham, "Experiments with analogy-x for software cost estimation," in *ASWEC '08*, 2008, pp. 229–238.
- [26] B. Nuseibeh, "To be and not to be: On managing inconsistency in software development," in *Proc. of 8th IEEE International Workshop on Software Specification & Design (IWSSD-8)*, 1996, pp. 164–169.
- [27] B. W. Boehm, C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. J. Reifer, and B. Steece, *Software Cost Estimation with Cocomo II*. Prentice Hall PTR, 2000.
- [28] M. Jorgensen, "A review of studies on expert estimation of software development effort," *Journal of Systems and Software*, vol. 70, pp. 37–60, 2004.
- [29] M. Jorgensen and T. Gruschke, "The impact of lessons-learned sessions on effort estimation and uncertainty assessments," *IEEE Trans. Softw. Eng.*, vol. 35, no. 3, pp. 368–383, May-June 2009.
- [30] M. Shepperd and G. F. Kadoda, "Comparing software prediction techniques using simulation," *IEEE Trans. Softw. Eng.*, vol. 27, no. 11, pp. 1014–1022, 2001.
- [31] H. Park and S. Baek, "An empirical validation of a neural network model for software effort estimation," *Expert Syst. Appl.*, vol. 35, no. 3, pp. 929–937, 2008.
- [32] A. Venkatachalam, "Software cost estimation using artificial neural networks," in *International joint conference on neural networks*, 1993, pp. 987–990.
- [33] G. Wittig and G. Finnie, "Estimating software development effort with connectionist models," *Information and Software Technology*, vol. 39, no. 7, pp. 469–476, 1997.
- [34] Y.-C. H. Arthur Earl Bryson, *Applied optimal control: Optimization, estimation, and control*. New York: Hemisphere Pub. Corp., 1969.
- [35] D. E. Rumelhart, L. McClelland, James, and the PDP Research Group, *Parallel distributed processing: explorations in the microstructure vol. 2*. MIT Press, 1986.
- [36] B. Kitchenham and E. Mendes, "Why comparative effort prediction studies may be invalid," in *PROMISE '09*. ACM, 2009, pp. 1–5.
- [37] 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.
- [38] P. N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Addison Wesley, 2005.

- [39] J. C. Gower and P. Legendre, "Metric and euclidean properties of dissimilarity coefficients," *Journal of Classification*, vol. 3, 1986.
- [40] J. Li, G. Ruhe, A. Al-Emran, and M. M. Richter, "A flexible method for software effort estimation by analogy," *Empirical Softw. Engg.*, vol. 12, no. 1, pp. 65–106, 2007.
- [41] C. Chang, "Finding prototypes for nearest neighbor classifiers," *IEEE Trans. on Computers*, pp. 1179–1185, 1974.
- [42] J. Gama and C. Pinto, "Discretization from data streams: applications to histograms and data mining," in *Symposium on Applied Computing*, 2006, pp. 662–667.
- [43] U. M. Fayyad and I. H. Irani, "Multi-interval discretization of continuous-valued attributes for classification learning," in *International Joint Conference on A. I.*, 1993, pp. 1022–1027.
- [44] Y. Yang and G. I. Webb, "A comparative study of discretization methods for naive-bayes classifiers," in *Pacific Rim Knowledge Acquisition Workshop*, 2002, pp. 159–173.
- [45] E. Frank, M. Hall, and B. Pfahringer, "Locally weighted naive bayes," in *Proceedings of the Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann, 2003, pp. 249–256.
- [46] L. Angelis and I. Stamelos, "A simulation tool for efficient analogy based cost estimation," *Emp. Softw. Engg.*, vol. 5, pp. 35–68, 2000.
- [47] J. R. Quinlan, "Learning with Continuous Classes," in *Joint Conference on A. I.*, 1992, pp. 343–348.
- [48] Y. Li, M. Xie, and G. T., "A study of the non-linear adjustment for analogy based software cost estimation," *Emp. Softw. Engg.*, pp. 603–643, 2009.
- [49] U. Lipowezky, "Selection of the optimal prototype subset for 1-*nn* classification," *Pattern Recog. Letters*, vol. 19, pp. 907–918, 1998.
- [50] C. Kirsopp and M. Shepperd, "Making inferences with small numbers of training sets," *Software, IEE Proceedings -*, vol. 149, no. 5, pp. 123 – 130, Oct. 2002.
- [51] I. Myrtveit, E. Stensrud, and M. Shepperd, "Reliability and validity in comparative studies of software prediction models," *IEEE Trans. Softw. Engg.*, vol. 31, no. 5, pp. 380–391, May 2005.
- [52] D. Beeferman and A. Berger, "Agglomerative clustering of a search engine query log," in *Knowledge Discovery and Data Mining*, pp. 407–416, 2000.
- [53] S. Guha, R. Rastogi, and K. S. Cure, "An efficient clustering algorithm for large databases," *ACM SIGMOD International Conference on Management of Data*, vol. pages, pp. 73–84, 1998.
- [54] M. B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein, "Cluster analysis and display of genome-wide expression patterns," *Proc. of the National Academy of Science*, vol. 95, pp. 14863–14868, 1998.
- [55] M. Steinbach, G. Karypis, and V. Kumar, "A comparison of document clustering techniques," in *KDD Workshop on Text Mining'10*.
- [56] E. P. Kasten and P. K. McKinley, "Meso: Supporting online decision making in autonomic computing systems," *IEEE Trans. on Knowl. and Data Engg.*, vol. 19, no. 4, pp. 485–499, 2007.
- [57] R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufman, 1992, ISBN: 1558602380.
- [58] J. Klijnen, "Sensitivity analysis and related analyses: a survey of statistical techniques," *Journal Statistical Computation and Simulation*, vol. 57, no. 1–4, pp. 111–142, 1997.
- [59] T. Menzies and S. Goss, "Vague models and their implications for the kbs design cycle," in *PKAW '96: Technical Report*, 1996, available from <http://bit.ly/gukExS>.
- [60] M. Shepperd, "Personnel communication on the value of different evaluation criteria," 2007.
- [61] S. Chulani, B. Boehm, and B. Steece, "Bayesian analysis of empirical software engineering cost models," *IEEE Trans. Softw. Engg.*, vol. 25, no. 4, pp. 573–583, 1999.
- [62] G. Boetticher, "When will it be done? the 300 billion dollar question, machine learner answers," *IEEE Intelligent Systems*, 2003.
- [63] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit, "A simulation study of the model evaluation criterion mmre," *IEEE Trans. Softw. Engg.*, vol. 29, no. 11, pp. 985 – 995, November 2003.
- [64] E. Alpaydin, *Introduction to Machine Learning*. MIT Press, 2004.
- [65] C. Robson, "Real world research: a resource for social scientists and practitioner-researchers," *Blackwell Publisher Ltd*, 2002.
- [66] Y. Wang, Q. Song, S. MacDonell, M. Shepperd, and J. Shen, "Integrate the gm(1,1) and verhulst models to predict software stage-effort," *IEEE Trans. on Systems*, vol. 39, pp. 647 – 658, 2009.
- [67] L. C. Briand, K. El Emam, D. Surmann, I. Wiecezorek, and K. D. Maxwell, "An assessment and comparison of common software cost estimation modeling techniques," in *ICSE '99: International conference on Software engineering*, 1999, pp. 313–322.

- [68] D. Milic and C. Wohlin, "Distribution patterns of effort estimations," in *Euromicro*, 2004.

- [69] A. Bakir, B. Turhan, and A. Bener, "A new perspective on data homogeneity in software cost estimation: A study in the embedded systems domain," *Software Quality Journal*, 2009. [Online]. Available: <http://bit.ly/fp4b9N>



Ekrem Kocaguneli holds an MSc and BS degrees in Computer Engineering from Bogazici University. Ekrem is a PhD candidate in CSEE at West Virginia University. His main research interests are software effort estimation, artificial intelligence applications in empirical software engineering and intelligent tools to aid software processes. He is currently a student member of IEEE and ACM.



Tim Menzies (PhD, UNSE) is an Assoc. Prof. in CSEE at WVU and the author of over 200 referred publications. At WVU, he has been a lead researcher on projects for NSF, NIJ, DoD, NASA's Office of Safety and Mission Assurance, as well as SBIRs and STTRs with private companies. He teaches data mining and artificial intelligence. Tim is the co-founder of the PROMISE conference series devoted to reproducible experiments in software engineering. In 2012, he will be the co-chair of the program committee for the IEEE Automated Software Engineering conference.



Ayse Basar Bener (PhD, LSE) is currently a Prof. in the Ted Rogers School of Information Technology Management at Ryerson University. Prior to joining Ryerson Dr. Bener founded and directed Software Research Lab (Softlab) at Bogazici University where many industry funded research projects were undertaken locally and globally. Her main research area is empirical software engineering: software measurement, software economics, and software quality. She mainly tackles the problem of decision making under uncertainty by using machine learning methods to build predictive models, cognitive science to model human behaviour and game theoretic models to determine strategies. She has more than 100 publications in these fields. She is a member of IEEE, ACM, and AAAI.



Jacky W. Keung (PhD, UNSW) is an Asst. Prof. in the Department of Computing at the Hong Kong Polytechnic University. Prior to joining PolyU, he was a Senior Research Scientist in the Software Engineering Research Group at NICTA. He also holds an academic position in the School of Computer Science and Engineering at the University of New South Wales. His current research interests are in software engineering for cloud computing, machine learning, data-intensive pattern analysis, software measurement and its application to project management, cost estimation, quality control and risk management, as well as software process improvement. His research results have been published in top quality journals and conferences, including IEEE Transactions on Software Engineering. He is a member of the Australian Computer Society, and a member of the IEEE Computer Society.