Active Learning and Effort Estimation: Finding the Essential Content of Software Effort Estimation Data

Ekrem Kocaguneli, *Student Member*, *IEEE*, Tim Menzies, *Member*, *IEEE*, Jacky Keung, *Member*, *IEEE*, David Cok, *Member*, *IEEE*, and Ray Madachy, *Member*, *IEEE*

Abstract—*Background:* Do we always need complex methods for software effort estimation (SEE)? *Aim:* To characterize the essential content of SEE data, i.e., the least number of features and instances required to capture the information within SEE data. If the essential content is very small, then 1) the contained information must be very brief and 2) the value added of complex learning schemes must be minimal. *Method:* Our QUICK method computes the euclidean distance between rows (instances) and columns (features) of SEE data, then prunes synonyms (similar features) and outliers (distant instances), then assesses the reduced data by comparing predictions from 1) a simple learner using the reduced data and 2) a state-of-the-art learner (CART) using all data. Performance is measured using hold-out experiments and expressed in terms of mean and median MRE, MAR, PRED(25), MBRE, MIBRE, or MMER. *Results:* For 18 datasets, QUICK pruned 69 to 96 percent of the training data (median = 89 percent). $K = 1$ nearest neighbor predictions (in the reduced data) performed as well as CART's predictions (using all data). *Conclusion:* The essential content of some SEE datasets is very small. Complex estimation methods may be overelaborate for such datasets and can be simplified. We offer QUICK as an example of such a simpler SEE method.

Index Terms—Software cost estimation, active learning, analogy, k-NN

1 Introduction

Accurate software effort estimation (SEE) is needed for many business tasks such as project plans, iteration plans, budgets, investment analysis, pricing processes, and bidding rounds. Such estimates can be generated via *expert-based methods* that use human expertise (possibly augmented with process guidelines, checklists, and data) to generate predictions [1], [2]. Alternatively, *model-based methods* can summarize old data with data miners that make predictions about new projects [3], [4].

The SEE literature is particularly interested in the summarization of old data by various model-based methods. The SEE literature review of Jorgensen and Shepperd [5] reports that 61 percent of the selected studies deal with the introduction of new methods and their comparison to old ones. These methods range in complexity from: relatively simple nearest neighbor methods [6], to iterative dichotomization methods, e.g., classification and regression trees (CART) [7], to complex search-based methods that, say, use tabu search to set the parameters of support vector regression [8] or exponential-time genetic algorithms that select best features or instances [9], to intricate search for stability through heuristic rejection rules like COSEEKMO [10], to ensemble of multiple estimation methods [11]. The justification for the complexity of the introduced method is a critical yet mostly overlooked issue. We argue that the complexity of the learning method should be matched to the *essential content* of the data. Given a matrix of N instances and F features, the essential content is $N' * F'$, where N' and F' are subsets of the instances and features, respectively. The methods learned from N' and F' perform as well as those learned from N and F.

This paper reports a search for N' and F' using a novel method called QUICK. QUICK computes the euclidean distance between the rows (instances) of SEE datasets. That distance calculation is also performed between matrix columns (features) using a transposed copy of the matrix. QUICK then removes synonyms (features that are very close to other features) and outliers (rows that are very distant to the others). QUICK then reuses the distance calculations a final time to find estimates for test cases, using the nearest neighbor in the reduced space.

The more complex the estimation methods become, the more prone they become to operator error. This is a growing problem. Shepperd et al. [12] report that the dominant factor that predicts for method performance is *who operates the data miner* (and not which dataset is studied, and not which data miner is used). This is a very troubling result that suggests our sophisticated data mining methods are now so complex that they have become very troublesome and error-prone. This paper

- *E. Kocaguneli and T. Menzies are with the Lane Department of Computer Science and Electrical Engineering, West Virginia University. E-mail: kocaguneli@gmail.com, tim@menzies.us.*
- *J. Keung is with the Department of Computer Science, City University of Hong Kong. E-mail: jackykeung@gmail.com.*
- *D. Cok is with GrammaTech, Inc., Ithaca, NY. E-mail: dcok@grammatech.com.*
- *R. Madachy is with the Naval Postgraduate School, Monterey, CA 93943. E-mail: rjmadach@nps.edu.*

Manuscript received 15 Feb. 2012; revised 28 Aug. 2012; accepted 10 Dec. 2012; published online 20 Dec. 2012.
Recommended for acceptance by D. Sjøberg.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-2012-02-0035.
Digital Object Identifier no. 10.1109/TSE.2012.88.

TABLE 1
The Symbols Used in This Paper

Symbol	Meaning
D	Denotes a specific data set
D'	Denotes the reduced version of D by QUICK
D^T	Transposed version of D
N	Number of instances in D
N'	Number of *selected* instances by QUICK from D
F	Number of independent features in D
F'	Number of independent features by QUICK from D
P	Represents a project in D
$Feat$	Represents a feature in D
i, j	Subscripts used for enumeration
P_i, P_j	i^{th} and j^{th} projects of D, respectively
DM	An $N \times N$ distance matrix that keeps distances between all projects of D
$DM(i,j)$	The distance value between P_i and P_j
k	Number of analogies used for estimation
E_{NN}	Everyone's nearest-neighbor matrix, which shows the order of P's neighbors w.r.t. a distance measure
$E(k)$	$E(k)[i,j]$ is 1 for $i \neq j$ and $E_{NN} \leq k$, 0 otherwise
$E(1)$	Specific case of $E(k)$, where $E(1)[i,j]$ is 1 if j is i's closest neighbor, 0 otherwise
$E(1)[i,j]$	The cell of $E(1)$ that corresponds to i^{th} row and j^{th} column
$\mathrm{Pop}(j)$	Popularity of j: $\mathrm{Pop}(j) = \sum_{i=1}^{n} E(1)[i,j]$
n	Number of projects consecutively added to active pool, i.e. a subset of D, i.e. $n \leq N$
Δ	The difference between the best and the worst error of the last n instances.
$x_i, \hat{x}_i$	Actual and predicted effort values of P_i, respectively.
$Perf_i, Perf_i$	i^{th} and j^{th} performance measures
M_i, M_i	i^{th} and j^{th} estimation methods

argues that the complexity of the estimation methods is only necessary if the added value is worth it.

The rest of this paper is structured as follows: The symbols used in this manuscript are listed in Table 1. The related work is discussed in Section 2. Section 3 introduces QUICK and its execution on a toy example. Our methodology is explained in Section 4. Section 5 shows experimental results. Section 6 reports sanity check results on proprietary datasets and Section 7 discusses threats to validity. Section 8 is a discussion of this work from the perspective of industrial practitioners. Future work and conclusions are listed in Sections 9 and 10.

2 Related Work

2.1 Active Learning

In active learning, some heuristic (in our case, each row's *popularity* value) is used to sort instances from most interesting to least interesting. The data are then explored according to that sort order. Learning can terminate early if the results from all N instances are not better than from a subset of M instances, where $M < N$.

There is a wealth of active learning studies in machine learning literature. For example, Dasgupta [13] seeks generalizability guarantees in active learning. He used a greedy active learning heuristic and showed that it is able to deliver performance values as good as any other heuristic in terms of reducing the number of required labels [13]. QUICK also uses a novel heuristic, i.e., the popularity of instances, so as to decide which instances to label first. Furthermore, similar to the active learning strategy provided by Dasgupta, QUICK's performance is comparable to that of supervised learners. Balcan et al. [14] show that active learning provides the same performance as a supervised learner with substantially smaller sample sizes. Although applied on a different type of datasets (i.e., SEE datasets) and using a different heuristic than Balcan et al.'s work, QUICK also shares the observed results of an active learning solution, e.g., similar performance to supervised learners with substantially smaller sample sizes. Wallace et al. [15] used active learning for a practical application example. They proposed a citation screening method based on active learning augmented with a priori expert knowledge. QUICK differs from that method in the sense that it does not use any a priori knowledge. However, the similarity arises from the fact that both methods use dataset properties to provide an ordering of the instances to be labeled.

In software engineering, the practical application exemplars of active learning can be found in software testing [16], [17]. In Bowring et al.'s [16] study, active learning is used to augment learners for automatic classification of program behavior. Bowring et al. show that learners augmented with active learning yield a significant reduction in data labeling effort and they can generate comparable results to those of supervised learning. The similarity between Bowring et al.'s work to our research is the fact that in both studies active learning is used as a means to reduce the data labeling effort without decreasing the estimation performance in comparison to supervised learners that use all the labels available. The difference between the two studies is that Bowring et al. used the labels as provided by the active learning algorithm to feed learners (e.g., Markov models), whereas in our case, the estimation phase coming from a learner (1 nearest neighbor) is a built-in part of QUICK. Xie and Notkin [17] used human inspection as an active learning strategy for effective test generation and specification inference. In their experiments, the number of tests selected for human inspection was feasible; the direct implication is that labeling required significantly less effort than screening all single test cases. In our study, QUICK does not require human inspection in its execution, yet its use is similar to Xie and Notkin's work, i.e., to reduce the effort of labeling by reducing the instances to be labeled. Hassan and Xie [18] list active learning as part of the future of software engineering data mining, and this paper proposes a novel method in this direction.

2.2 Instance and Feature Subset Selection (FSS)

One other way to view QUICK's outlier pruning is as an *unsupervised instance selection* algorithm. *Supervised algorithms* require instance labels, while *unsupervised ones* execute on unlabeled data. *Instance selection* algorithms generate *prototypes*, i.e., a subset of the data which best represents the predictive properties of the whole data. A standard result in instance selection is that most of the rows in a matrix of data can be removed without damaging the predictive power of rules learned from the remaining data. For example, Chang's [19] prototype generators explored three datasets A, B, C of size 51,415,066 instances, which were reduced down to 7, 9, and 9 percent of the original data, respectively. For example, Li et al. [9] use a genetic algorithm to search for the subset of instances that yield the best estimates. The TEAK algorithm of Kocaguneli et al. [20] clustered instances, then selected clusters with low class

label variance. The difference of QUICK from supervised instance selection methods is that it does not require detailed costing data on all instances, whereas supervised methods require project "labels" on all examples.

Another way to view QUICK's synonym pruning is as an *unsupervised* FSS algorithm. It is well known that selecting a subset of the SEE dataset features can improve the estimation performance. For example, Lum et al. [21] capture and report the best practices in SEE. One of the fundamental suggestions among the best practices is FSS. They show that both manual and supervised FSS methods improve estimation performance [21]. Other FSS examples are the stepwise regression (SWR) [22] and principal component analysis (PCA) [11], [23]. The common property of the aforementioned FSS algorithms, except for PCA, is that they are *supervised*. *Supervised algorithms* require the instance labels (i.e., dependent variables). QUICK, on the other hand, is an *unsupervised* FSS algorithm. Unlike supervised algorithms, QUICK can execute without labels, which removes the necessity of label collection prior to FSS. In comparison to PCA, QUICK's method of finding feature subsets is easier to implement and understand, in particular for those without a considerable machine learning background. QUICK only requires knowledge of normalization of an array of instances and the calculation of the euclidean distance. PCA, on the other hand, requires the user to know the concepts of correlation between the features and the orthogonal transformation [24]. Unlike QUICK, PCA's output is not a subset of the individual features that a user sees on the datasets, but rather a new set of—less correlated—features (principal components), which are linear combinations of the original features.

3 QUICK

QUICK is an active learning method that assists in reducing the complexity of data interpretation by identifying the essential content of SEE datasets. QUICK works as follows:

1. Group rows and columns by their similarity,
2. Discard redundant columns (synonyms) that are too similar,
3. Discard outlier rows (outliers) that are too distant,
4. In the remaining data, generate an estimate from the nearest example.

The following sections provide detailed information on these steps.

3.1 Pruning Synonyms

Synonyms are features closely associated with each other. QUICK removes such redundant features as follows:

Step 1: Transpose dataset matrix. This step may or may not be necessary depending on how the initial dataset is stored. However, rows of SEE datasets usually represent past project instances, whereas the columns represent the features defining these projects. When such a matrix is transposed, the project instances are represented by columns and project features are represented by the rows. Note that columns are normalized to the 0-1 interval before transposing to remove the superfluous effect of large numbers in the next step.

Step 2: Generate distance matrices. For the transposed dataset D^T of F instances, the associated distance matrix (DM) is an $F \times F$ matrix keeping the distances between every feature pair according to euclidean distance function. For example, a cell located at the ith row and jth column ($DM(i,j)$) keeps the distance between the ith and jth features (diagonal cells ($DM(i,i)$) are ignored).

Step 3: Generate E_{NN} and $E(1)$ matrices. $E_{NN}[i,j]$ shows the neighbor rank of "j" w.r.t. "i," e.g., if "j" is "i's" third nearest neighbor, then $E_{NN}[i,j] = 3$. The trivial case where $i = j$ is ignored, i.e., an instance's nearest neighbor does not include itself. The $E(k)$ matrix is defined as follows: If $i \neq j$ and $E_{NN}[i,j] \leq k$, then $E(k)[i,j] = 1$; otherwise, $E(k)[i,j] = 0$. In synonym pruning, we want to select the unique features without any nearest neighbors. For that purpose, we start with $k = 1$; hence, E(1) identifies the features that have at least another nearest neighbor and the ones without any nearest neighbor. The features that appear as one of the k-closest neighbors of another feature are said to be popular. The "popularity index" (or simply "popularity") of feature "j," "$\mathrm{Pop}(Feat_j)$," is defined to be $\mathrm{Pop}(Feat_j) = \sum_{i=1}^{n} E(1)[i,j]$, i.e., how often the "$j$th" feature is some other feature's nearest neighbor.

Step 4: Calculate the popularity index based on $E(1)$ and select nonpopular features. Nonpopular features are the ones that have a popularity of zero, i.e., $\mathrm{Pop}(Feat_i) = 0$.

3.2 Pruning Outliers

Outlier pruning is similar to synonym pruning—with certain important differences: With synonym pruning, we *transpose* the data to find the distances between "rows" (which in the transposed data are features). Then, we count the *popularity* of each feature and delete the *popular* features (these are the features that needlessly repeated the information found in other features). With outlier pruning, we *do not transpose* the data before finding the distances between rows. Then, we count the *popularity* of each row and delete the *unpopular* rows (the instances that are most distant from the others). Note that the dataset used to prune outliers contains only the selected features of the previous phase. Also, note that the terms feature and variable will be used interchangeably from now on. Following is the steps of the outlier pruning phase:

Step 1: Generate distance matrices. For a dataset D of size N, the associated DM is an $N \times N$ matrix whose cell located at row i and column j ($DM(i,j)$) keeps the distance between the ith and jth instances of D. The cells on the diagonal ($DM(i,i)$) are ignored. Note that current D comes from the phase of synonym pruning; hence, it only has the selected features.

Step 2: Generate E_{NN} and $E(1)$ matrices. $E_{NN}[i,j]$ shows the neighbor rank of "j" w.r.t. "i." Similar to the step of synonym pruning, if "j" is "i's" third nearest neighbor, then $E_{NN}[i,j] = 3$. Again, the trivial case of $i = j$ is ignored (nearest neighbor does not include itself). The $E(k)$ matrix has exactly the same definition as the one in synonym pruning phase: If $i \neq j$ and $E_{NN}[i,j] \leq k$, then $E(k)[i,j] = 1$; otherwise, $E(k)[i,j] = 0$. In this study, the nearest neighbor-based ABE is considered, i.e., we use $k = 1$; hence, E(1) describes just the single nearest neighbor. All instances that appear as one of the k-closest neighbors of another instance are defined to be popular. The "popularity index" (or simply "popularity") of instance "j," "$\mathrm{Pop}(j)$," is defined to

TABLE 2
The Percentage of the Popular Instances (to Dataset Size) Useful for Prediction in a Closest Neighbor Setting

Dataset	% popular instances
kemerer	80
telecom1	78
nasa93_center_1	75
cocomo81s	73
finnish	71
cocomo81e	68
cocomo81	65
nasa93_center_2	65
nasa93_center_5	64
nasa93	63
cocomo81o	63
sdr	63
desharnaisL1	61
desharnaisL2	60
desharnaisL3	60
miyazaki94	58
desharnais	57
albrecht	54
maxwell	13

Only the instances that are closest neighbors of another instance are said to be popular.

be $\mathrm{Pop}(j) = \sum_{i=1}^{n} E(1)[i,j]$, i.e., how often "$j$" is someone else's nearest neighbor.

Step 3: Calculate the popularity index based on $E(1)$ and determine the sort order for labeling. As shown in Table 2, the popular instances j with $\mathrm{Pop}(j) \geq 1$ (equivalently, $E(1)[i,j]$ is 1 for some i) have a median percentage of 63 percent among all datasets, i.e., more than one-third of the data is unpopular with $\mathrm{Pop}(j) = 0$. Following this observation, we speculated that if we label data in order of its popularity, then we would be labeling the most important projects *first*.

Step 4: Find stopping point and halt. The notion behind instance selection is to reach conclusions using fewer instances. To test that, QUICK labels some instances (adding them to the active pool) then stops; we have defined the following stopping rules:

1. All instances with $\mathrm{Pop}(j) \geq 1$ are exhausted.
2. Or there is no estimation accuracy improvement in the active pool for n consecutive times.
3. Or the Δ between the best and the worst error of the last n instances in the active pool is very small.

For the error measure in point #3, we used "magnitude of relative error (MRE)," i.e., the magnitude of relative error $(abs(actual - predicted)/actual)$. MRE is only one of many possible error measures. As shown below, even though we guide the search using only MRE, the resulting estimations score very well across a wide range of error measures. In our experiments, we used $n = 3$ and $\Delta < 0.1$. The selection of (n, Δ) values is based on our engineering judgment. The sensitivity analysis of trying different values of (n, Δ) can be promising future work.

Retaining policy of instances is different from that of features. Instances with high popularity are retained, whereas for the features, high popularity is a reason to be discarded. Distant instances without any neighbors are likely to be outliers, and we expect an essential set of instances with high popularity to capture the essential content of a dataset. Distant features with low popularity are expected to reflect a different view of the data, whereas features with high popularity may be sharing information with their neighboring features.

Project	$Feat_1$	$Feat_2$	$Feat_3$	Effort
P_1	1	2	20	3
P_2	2	4	10	4
P_3	3	6	40	7

Fig. 1. Three projects defined by three independent features/variables and a dependent variable.

	P_1	P_2	P_3
$Feat_1$	0.0	0.5	1
$Feat_2$	0.0	0.5	1
$Feat_3$	0.3	0.0	1

Fig. 2. Matrix after normalizing and transposing D.

3.3 Examples

This section offers a small example of QUICK. Assume that the training set of the example consists of three instances/projects: P_1, P_2, and P_3. Also assume that these projects have one dependent and three independent features. Our dataset would look like that shown in Fig. 1.

3.3.1 Synonym Pruning

Step 1: Transpose dataset matrix. After normalization to the 0-1 interval then transposing our dataset, the resulting matrix would look like that shown in Fig. 2.

Step 2: Generate DMs. The DM keeps the euclidean distance between features. Fig. 2 is used to calculate the DM in Fig. 3.

Step 3: Generate E_{NN} and $E(1)$ matrices. According to the DM in Fig. 3, the resulting $E_{NN}[i,j]$ in Fig. 4 shows the neighbor ranks of features. Using E_{NN}, we calculate the $E(1)$ matrix (Fig. 5) that identifies the features with at least another nearest neighbor.

Step 4: Calculate the popularity index based on $E(1)$ and select nonpopular features. Popularity of a feature is the total of $E(1)$'s columns (see the summation in Fig. 5). *Nonpopular* features are the ones with zero popularity. In this toy example, we only select $Feat_3$ because it is the only column with zero popularity.

3.3.2 Outlier Removal and Estimation

In this phase, QUICK continues with only the selected features. Now, our dataset looks like that shown in Fig. 6.

Step 1. The first step of QUICK in this phase is to *build the DM*. Since projects are described by a single attribute $Feat_3$, the euclidean distance between two projects will be the difference of the normalized $Feat_3$ values. Fig. 7 shows the resulting *DM*.

Step 2. Creating the E_{NN} matrix based on the DM is the second step. As we are creating the E_{NN} matrix, we traverse the DM row by row and label the instances depending on their distance order: The closest neighbor is labeled 1, the second closest neighbor is labeled 2, and so on. Note that diagonal entries with the distance values of 0 are ignored as they represent the distance of the instance to itself, not to a neighbor. After this traversal, the resulting E_{NN} is given in Fig. 8.

Step 3: Calculating the popularity index based on E_{NN} and determining the labeling order. Remember from the previous section that $E(1)$ is generated from E_{NN}: $E(1)[i,j] = 1$ if

	$Feat_1$	$Feat_2$	$Feat_3$
$Feat_1$	na	0.0	0.6
$Feat_2$	0.0	na	0.6
$Feat_3$	0.6	0.6	na

Fig. 3. DM for features.

	$Feat_1$	$Feat_2$	$Feat_3$
$Feat_1$	na	1	2
$Feat_2$	1	na	2
$Feat_3$	1	1	na

Fig. 4. The E_{NN} matrix for features, resulting from the DM of Fig. 3. Diagonal cells are ignored.

		$Feat_1$	$Feat_2$	$Feat_3$
	$Feat_1$	0	1	0
	$Feat_2$	1	0	0
+	$Feat_3$	1	1	0
	$Popularity:$	2	1	0

Fig. 5. Popularity of the features. Popularity is the sum of the $E(1)$ matrix columns.

Project	$Feat_3$	Effort
P_1	20	3
P_2	10	4
P_3	40	7

Fig. 6. Three projects defined by $Feat_3$ (say, KLOC) and effort (say, in staff months).

	P_1	P_2	P_3
P_1	0	0.3	0.7
P_2	0.3	0	1
P_3	0.7	1	0

Fig. 7. The DM of the projects P_1, P_2, and P_3.

$E_{NN}[i,j] = 1$; otherwise, $E(1)[i,j] = 0$. The popularity index associated with each instance is then calculated by summing the values in every column (i.e., the sum of the first column is the popularity index of the first instance, the sum of the second column is the popularity index of the second instance, and so forth). The $E(1)$ matrix and the popularity indices of our example are given in Fig. 9. Note that $E(k)$ matrices are not necessarily symmetric; see $E(1)$ of Fig. 9.

Step 4: Finding the stopping point. The change in the active pool for the toy example is shown in Fig. 10. In an actual setting, we only move from $Round_i$ to $Round_{i+1}$ if the stopping rules do not fire.

	P_1	P_2	P_3
P_1	na	1	2
P_2	1	na	2
P_3	1	2	na

Fig. 8. The E_{NN} matrix resulting from the DM of Fig. 7. Diagonal cells are ignored.

		P_1	P_2	P_3
	P_1	0	1	0
	P_2	1	0	0
+	P_3	1	0	0
	$Popularity:$	2	1	0

Fig. 9. Popularity is the sum of $E(1)$'s columns.

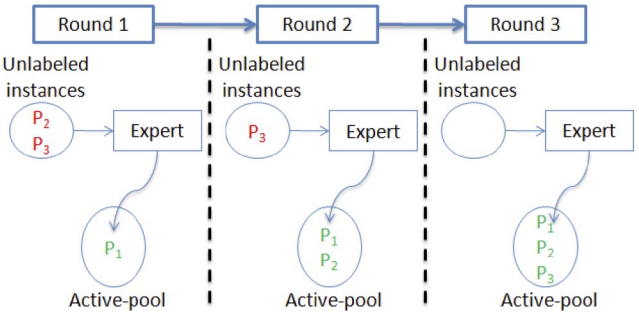


Fig. 10. The change in the active pool for the toy example. In an actual setting, the transition between $Round_i$ to $Round_{i+1}$ is governed by the stopping rules.

4 Methodology

4.1 Datasets

Our study uses a total of 18 datasets (listed in Table 3), *COCOMO* datasets (cocomo*, nasa*), which are collected with the COCOMO approach [25]. Three of these datasets (nasa93_center_1, nasa93_center_2, nasa93_center_5) come from different NASA development centers around the United States. Three other datasets are mostly from Southern California aerospace companies (cocomo81e, cocomo81o, cocomo81s). An important note here is the handling of nominal values of the datasets that were collected according to the COCOMO method. The COCOMO datasets can have nominal values such as *"low," "high," "very-high,"* and so on, and these values also have corresponding numeric values as explained by Boehm [25]. In our paper, we converted nominal values to their numeric equivalents. Another widely used dataset in SEE is the *desharnais* dataset that contains software projects from Canada. It is collected with that function points approach.

Be aware that the desharnais dataset has missing values for three projects. Standard ways of handling missing values are: 1) Throw away the projects with missing entries or 2) use imputation [24] to derive the missing values from complete projects. We opted for the latter option and used mean imputation for the missing values. *SDR* contains data from recent projects of various software companies in Turkey. SDR is collected by Softlab, the Bogazici University Software Engineering Research Laboratory [26], and it is one of the newest datasets used in this research. The *albrecht* dataset consists of projects completed by IBM in the 1970s, and details are given in [27]. The *finnish* dataset contains 40 projects from different companies and was collected by a single person in the 1990s. The two projects with missing

TABLE 3
The 681 Projects Used in This Study Come from 18 Datasets

Dataset	Features	Size	Description	Units
cocomo81	17	63	NASA projects	months
cocomo81e	17	28	Cocomo81 embedded projects	months
cocomo81o	17	24	Cocomo81 organic projects	months
cocomo81s	17	11	Cocomo81 semi-detached projects	months
nasa93	17	93	NASA projects	months
nasa93_center_1	17	12	Nasa93 projects from center 1	
nasa93_center_2	17	37	Nasa93 projects from center 2	months
nasa93_center_5	17	40	Nasa93 projects from center 5	months
desharnais	11	81	Canadian software projects	hours
desharnaisL1	11	46	Projects developed in Language1	hours
desharnaisL2	11	25	Projects developed in Language2	hours
desharnaisL3	11	10	Projects developed in Language3	hours
sdr	22	24	Turkish software projects	months
albrecht	7	24	Projects from IBM	months
finnish	8	38	Projects developed in Finland	hours
kemerer	7	15	Large business applications	months
maxwell	27	62	Banking projects in Finland	hours
miyazaki94	8	48	Projects from Japan	months
		681		

Indentation in column one denotes that the indented dataset is a subset of its nonindented parent.

values are omitted here; hence, we use 38 instances. More details can be found in [28]. *kemerer* is a relatively small dataset with 15 instances; details can be found in [29]. *maxwell* is another relatively new dataset (projects from the late 1990s to early 2000s) that comes from the finance domain and is composed of Finnish banking software projects. Details are given in [30]. *miyazaki* contains projects developed in COBOL. For details, see [31].

4.2 Algorithms

In this study, we use nearest neighbor and CART methods. We justify our selection of SEE methods as follows: Several papers conclude that CART and nearest neighbor methods are useful comparison algorithms for SEE. Walkerden and Jeffrey [32] endorse CART as a state-of-the-art SEE method. There are two recent studies published in the *IEEE Transactions on Softare Engineering* commenting on what effort estimation methods are the best [11], [33]. These studies assert that in terms of assessing new effort estimation methods, existing methods such as CART's regression tree generation may be more than adequate, e.g., Dejaeger et al. [33] found little evidence that learners more elaborate than CART offer significant value added. Our own results endorse the conclusions of Walkerden et al.: A study of 90 effort estimators that use all combinations of 10 preprocessors and 9 learners to build ensemble methods [11]. The preprocessors were normalization, various discretizers, and feature selectors. The learners included *k*-NN (with $k = 1$, $k = 5$), linear and SWR, CART, and neural nets. As might have been predicted by Shepperd and Kadoda [34], the ranking of the estimators varied across different datasets and the different accuracy estimators. However, we found a small group of 13 estimators (all variants of CART and *k*-NN aided with preprocessors) that were consistently the best performing methods. These methods were some combination of the following preprocessors and learners.

The preprocessors studied in [11] are *logging (log)* and *normalization (norm).* With the *norm* preprocessor, numeric values are normalized to a 0-1 interval using (1). The normalization is to ensure that no individual variable has greater influence than others:

$$normalized = \frac{(actualValue - min(allValues))}{(max(allValues) - min(allValues))}. \quad (1)$$

With the *log* preprocessor, all numerics are replaced with their natural logarithm value. This procedure minimizes the effects of the occasional very large numeric values.

The learners studied in [11] were: 1) *an iterative dichotomizer, CART*, 2) *an instance-based learner,* ABE0-kNN. Iterative dichotomizers like CART find the attribute that most divides the data such that the variance of each division is minimized. The algorithm then recurses on each division. Finally, the cost data of the instances in the leaf nodes are averaged to generate the estimate.

ABE0-kNN is our name for a very basic type of ABE that we derived from various ABE studies [9], [35], [36]. In *ABE0-kNN*, independent variables are first normalized to the 0-1 interval, then the distances between test and training instances were measured according to a euclidean distance function. Note that standard euclidean distance function requires numeric values in the datasets, which is valid in this research. However, it is suggested that a reader willing to use ABE0-*k*NN on datasets that contain nonnumeric values consider alternative solutions that adapt the euclidean distance function to handle nonnumeric values. This caution is also valid for logging and normalization. Following the previous step, k nearest neighbors are chosen from the training set and then their median cost value is returned as the estimate. We adopted a single k value in this study. *ABE0-1NN:* Only the closest analogous instance is chosen. The median of a single value is itself; therefore, the estimated value is the actual effort value of the closest neighbor.

The two preprocessors and the learners are combined to form two different algorithms: 1) log&ABE0-1NN and 2) norm&CART. We will use two different versions of *log&ABE0-1NN*: the one working on the so-called *active pool* (the pool that contains only the instances labeled by QUICK) and the one working on a training set with all instances labeled. For convenience, we will name the former *"activeNN"* and the latter *"passiveNN."* Note that the activeNN is our QUICK algorithm. Since we have only one CART-based algorithm (*norm&CART*), the learner name (CART) and the algorithm name (*norm&CART*) will be used interchangeably from now on. Note that the two preprocessors and two learners yield four possible combinations and we use only two of them. Elsewhere [37], we compared all four possible combinations (together with a total of 90 different methods) on a total of 20 datasets using seven different error measures. This comparison showed that *log&ABE0-1NN* and *norm&CART* perform better than the other two combinations; hence, we use them.

4.3 Experimental Design

Our experimental rig has three parts:

1. Generate *baseline* results. Apply CART and passiveNN on the entire training set.
2. Generate the *active learning* results. Run QUICK.
3. Compare the baseline results against results of QUICK.

The first part of the experimental rig aims at generating baseline results of successful supervised learners (CART and passiveNN) that have been shown to have high performance [11], [32], [33] (a detailed discussion on the choice of these learners can be found in Section 4.2). The baseline results will be used to compare the success of the proposed active learning method, QUICK. The second part of the experimentation serves to generate the estimates of QUICK for this comparison: QUICK is run as an active learning method (i.e., not all the labels are used) and its estimates are stored. The third part of the experimentation compares the estimates of QUICK against CART and passiveNN with respect to a number of error measures (see Section 4.4 for the details of the used error measures) evaluated according to an appropriate statistical test. The comparison performed in the third part of the experimentation shows whether QUICK is comparable to supervised learners such as CART and passiveNN, i.e., its purpose is to reveal whether the data reduction proposed by QUICK comes at the expense of higher error rates or whether QUICK performs as well as CART and passiveNN when finding the essential content of the data. The details of these steps are:

1. *Generate baseline results* by applying CART and passiveNN on the entire training set. The algorithms are run on the entire training set and their estimations are stored. We use 10-way cross validation:
   a. Randomize the order of instances in the dataset.
   b. Divide dataset into 10 bins.
   c. Choose 1 bin at a time as the test set and use the remaining bins as the training set.
   d. Repeat the previous step using each one of the 10 bins in turn as the test set.
2. *Generate the active learning results* by running QUICK. At each iteration, first the features are selected and the active pool is populated with training instances in the order of their popularity. Training instances outside the active pool are considered unlabeled and QUICK is only allowed to use instances in the pool. Train and test sets are generated by 10-way-cross validation.

   Before a training instance is placed in the active pool, an expert labels it, i.e., the costing data are collected. When the active pool only contains one instance, the estimates will all be the same. As the active pool is populated, QUICK has more labeled training instances to use.
3. *Compare baseline to active learning.* Once the execution of the algorithms is complete, the performances of QUICK, *passiveNN*, and CART are compared under different performance measures. The QUICK estimates used for comparison are the ones generated by the active pool at the stopping point. An important point to note here is that QUICK has no relation to the derivation of the baseline results for *passiveNN* and CART. It is true that QUICK makes use of *k*-NN methods in its execution; however, the derivation of baseline *passiveNN* and CART are separate procedures from the derivation of QUICK. In the comparison phase, the results of these separate procedures are compared. Another point to note is that QUICK only uses the selected features (from the first execution phase) to decide which instances are to be labeled and placed into the active pool.

4.4 Performance Measures

Performance measures comment on the success of a prediction. A variety of performance measures are used in the software engineering literature, none of which are endorsed across the whole community. Rather than entering into that debate, we assess our methods w.r.t. multiple measures.

Let x_i and $\hat{x}_i$ be the actual and predicted values for test instance i, respectively. The absolute residual (AR) is the difference between predicted and actual:

$$AR_i = |x_i - \hat{x}_i|. \tag{2}$$

The MRE measure is the ratio of the AR to the actual effort:

$$MRE_i = \frac{\mid x_i - \hat{x}_i \mid}{x_i} = \frac{AR_i}{x_i}. \tag{3}$$

A related measure is the magnitude of error relative (MER) to the estimate:

if Wilcoxon($Perf_i$, $Perf_j$, 95) says there is no statistical difference. **then**
  $tie_i = tie_i + 1$;
  $tie_j = tie_j + 1$;
else
  if better($Perf_i$, $Perf_j$) **then**
    $win_i = win_i + 1$
    $loss_j = loss_j + 1$
  else
    $win_j = win_j + 1$
    $loss_i = loss_i + 1$
  end if
end if

Fig. 11. Comparing algorithms (i, j) on performance $(Perf_i, Perf_j)$. The "better" predicate changes according to $Perf$, e.g., for MRE, "better" means lower values, whereas for PRED(25), "better" means higher values.

$$MER_i = \frac{\mid x_i - \hat{x}_i \mid}{\hat{x}_i} = \frac{AR_i}{\hat{x}_i}. \tag{4}$$

The overall average error of MRE can be derived as the mean or median magnitude of relative error measure (MMRE, or MdMRE, respectively) can be calculated as

$$MMRE = \frac{\sum_{i=1}^{n} MRE_i}{n}, \tag{5}$$

$$MdMRE = median(allMRE_i). \tag{6}$$

PRED(25) is the percentage of predictions falling within 25 percent of the actual values and can be expressed as

$$PRED(25) = \frac{100}{N} \sum_{i=1}^{N} \begin{cases} 1, & \text{if } MRE_i \leq \frac{25}{100} \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

For example, PRED(25) = 50 percent implies that half of the estimates fall within 25 percent of the actual values [38].

The mean balanced relative error (MBRE) and the mean inverted balanced relative error (MIBRE) are defined as follows:

$$MBRE_i = \frac{|\hat{x}_i - x_i|}{min(\hat{x}_i, x_i)} \quad MIBRE_i = \frac{|\hat{x}_i - x_i|}{max(\hat{x}_i, x_i)}. \tag{8}$$

Performance measures should be supplemented with appropriate statistical checks (otherwise, they may lead to biased or even false conclusions [39]). We use the Wilcoxon signed-rank test (95 percent confidence). Wilcoxon is more robust than the Student's t-test as it compares the sums of ranks, unlike student's t-test, which may introduce spurious findings as a result of outliers in the given datasets. Nonparametric tests like Wilcoxon are also useful if it is not clear that the underlying distributions are Gaussian [40].

Using Wilcoxon, we generate win, tie, and $loss$ statistics via the procedure in Fig. 11. We first check if two distributions i, j are statistically different; otherwise, we increment tie_i and tie_j. If the distributions are statistically different, we update win_i, win_j and $loss_i, loss_j$ after comparing the performance measures.

Note that, to make the case for QUICK, we do not need to show that an active learner generates *better* estimates.

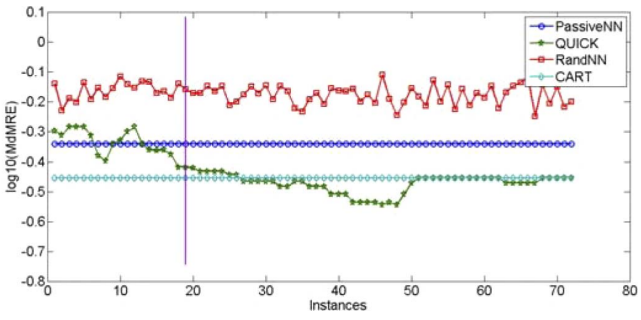


Fig. 12. Sample plot of a representative (in this case, desharnais) dataset showing the stopping point (the line parallel to the y-axis at $x = 19$) and MdMRE values (logged with base 10 for easier visualization). Note that QUICK uses only the fou selectedr features of desharnis dataset, whereas other methods use all 10 of the features.

Rather, we can recommend QUICK if its estimates are *no worse* than other approaches (caveat: just as long as it does so using less project data). Hence, we report the following experiments in terms of *losses* since if two methods M_1 and M_2 lose at equal frequency, then there is no case where M_i is overall worse than M_j.

5 Results

5.1 Estimation Performance

Fig. 12 shows a sample plot for a representative dataset (shown is the *desharnais* dataset). It is the result of QUICK's synonym pruning (four features selected for desharnais) followed by labeling N' instances in decreasing popularity. Following is the reading in Fig. 12:

- The Y-axis is the logged MdMRE error measures. The smaller the value, the better the performance.
- The line with star dots shows the error seen when the ith instance was estimated using the labels $1 \ldots (i-1)$.
- The horizontal lines show the errors seen when estimates were generated using all the data (either from CART or *passiveNN*).
- The vertical line shows the point where QUICK advised that labeling can stop (i.e., N').
- The square-dotted line shows *randNN*, which is the result of picking any random instance from the training set and using its effort value as the estimate.

Three observations of importance from Fig. 12 are the 1) *the reduction in the number of instances required*, 2) *the reduction in the number of features*, and 3) *the estimation error*. With a fraction of the instances and the features of the original dataset, QUICK is able to get as low error rates as CART and passiveNN, which both use the entire dataset. Furthermore, we have seen that the observed effect of QUICK is not random, as *randNN* has a much higher prediction error. To observe QUICK's performance comparison on other datasets, we will not use Fig. 12 any more due to two reasons: 1) Repeating Fig. 12 for *7 error measures* $\times$ *17 datasets* would not fit into the current draft; 2) more importantly, Fig. 12 does not tell whether differences are significant, e.g., see Tables 5 and 6 which show that the performance differences of QUICK compared to passiveNN and CART are not significant for the desharnais dataset. Therefore, we provide a summary analysis in the following sections.

TABLE 4
The Number of Selected Features (F') and Selected Instances (N') of a Dataset D with N Instances and F Independent Features

Dataset	N	N'	$\frac{N'}{N}$	F	F'	$\frac{F'}{F}$	$\frac{N'*F'}{N*F}$
nasa93	93	21	23 %	16	3	19 %	4 %
cocomo81	63	11	17 %	16	6	38 %	7 %
cocomo81o	24	13	54 %	16	2	13 %	7 %
maxwell	62	10	16 %	26	13	50 %	8 %
kemerer	15	4	27 %	6	2	33 %	9 %
desharnais	81	19	23 %	10	4	40 %	9 %
miyazaki94	48	17	35 %	7	2	29 %	10 %
desharnaisL1	46	12	26 %	10	4	40 %	10 %
nasa93_center_5	39	11	28 %	16	6	38 %	11 %
nasa93_center_2	37	16	43 %	16	4	25 %	11 %
cocomo81e	28	9	32 %	16	7	44 %	14 %
desharnaisL2	25	6	24 %	10	6	60 %	14 %
sdr	24	10	42 %	21	8	38 %	16 %
desharnaisL3	10	6	60 %	10	3	30 %	18 %
nasa93_center_1	12	4	33 %	16	9	56 %	19 %
cocomo81s	11	7	64 %	16	5	31 %	20 %
finnish	38	17	45 %	7	4	57 %	26 %
albrecht	24	9	38 %	6	5	83 %	31 %

This table is sorted by the last column.

5.2 Reduction in Sample and Feature Size

Table 4 shows reduction results from all 18 datasets used in this study. The N column shows the size of the data and the N' column shows how much of that data was labeled by QUICK. The $\frac{N'}{N}$ column expresses the percentage ratio of these two numbers. In a similar fashion, F shows the number of *independent features* for each dataset, whereas F' shows the number of selected features by QUICK. The $\frac{F'}{F}$ ratio expresses the percentage of selected features to the total number of the features. The important observation of Table 4 is that given N projects and F features, it is neither necessary to collect detailed costing details on 100 percent of N nor is it necessary to use all the features. For nearly half the datasets studied here, labeling around one-third of N would suffice (the median of $\frac{N'}{N}$ for 18 datasets is 32.5 percent). There is a similar scenario for the amount of features required. QUICK selects around one-third of F for half the datasets. The median value of $\frac{F'}{F}$ for 18 datasets is 38 percent.

The combined effect of synonym and outlier pruning becomes clearer when we look at the last column of Table 4. Assuming that a dataset D of N instances and F independent features is defined as an N-by-F matrix, the reduced dataset D' is a matrix of size N' by F'. The last column shows the total reduction provided by QUICK in the form of a ratio: $\frac{N'*F'}{N*F}$. The rows in Table 4 are sorted according to this ratio. Note that the maximum size requirement (albrecht dataset) is 32 percent of the original dataset and with QUICK we can go as low as only 4 percent of the actual dataset size (nasa93).

From the above result, we are able to conclude that the reduced datasets (D') proposed for discussion adequately represent all the project cases available in the dataset, given the resultant differences are statistically insignificant. Our only explanation for the success of such a method (using such tiny regions of the original data) is that the essential content of effort datasets *can be localized in just a few rows and just a few columns*. Then: 1) Elaborate estimation methods

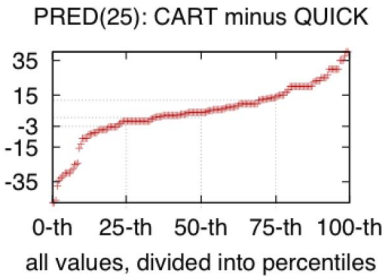


Fig. 13. CART minus QUICK values for PRED(25) in 18 datasets. Negative values mean that QUICK outperformed CART. The median is only 2 percent with a small interquartile range (75th − 25th percentile) of 15 percent.

will learn little more than simpler ones (since there is so little to find); 2) we can recommend simpler methods (e.g., QUICK).

5.3 Comparison QUICK versus CART

Fig. 13 shows the PRED(25) difference between CART (using all the data) and QUICK (using just a subset of the data). The difference is calculated as *PRED(25) of CART minus PRED(25) of QUICK*. Hence, a *negative* value indicates that QUICK offers better PRED(25) estimates than CART. The left-hand side (starting from the value of -35) shows QUICK is better than CART, whereas in other cases CART outperformed QUICK (see the right-hand side until the value of $+35$).

TABLE 5
QUICK (on the Reduced Dataset) versus passiveNN (on the Whole Dataset) w.r.t. Number of Losses, i.e., *Lower* Values Are *Better*

Dataset	Method	MMRE	MAR	Pred(25)	MdMRE	MBRE	MIBRE	MMER	SUM OF LOSSES
cocomo81	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
cocomo81e	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
cocomo81o	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
cocomo81s	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
desharnais	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
desharnaisL1	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
desharnaisL2	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
desharnaisL3	passiveNN	0	0	0	0	0	0	0	0
	QUICK	1	0	1	1	1	1	1	6
nasa93	passiveNN	0	0	0	0	1	1	0	2
	QUICK	0	0	0	0	0	0	0	0
nasa93_center_1	passiveNN	1	0	1	1	1	1	1	6
	QUICK	0	0	0	0	0	0	0	0
nasa93_center_2	passiveNN	1	1	1	1	1	1	1	7
	QUICK	0	0	0	0	0	0	0	0
nasa93_center_5	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
sdr	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
albrecht	passiveNN	1	1	1	1	1	1	0	6
	QUICK	0	0	0	0	0	0	0	0
finnish	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
kemerer	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
maxwell	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	1	1	1	3
miyazaki94	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0

Highlighted rows show datasets where QUICK performs worse than passiveNN in the majority case (4 out of 7, or more).

TABLE 6
QUICK (on the Reduced Dataset) versus CART (on the Whole Dataset) w.r.t. Number of Losses, i.e., *Lower* Values Are *Better*

Dataset	Method	MMRE	MAR	Pred(25)	MdMRE	MBRE	MIBRE	MMER	SUM OF LOSSES
cocomo81	CART	0	0	0	0	0	0	0	0
	QUICK	1	1	1	1	1	1	1	7
cocomo81e	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
cocomo81o	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
cocomo81s	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
desharnais	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
desharnaisL1	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
desharnaisL2	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
desharnaisL3	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	1	1	1	3
nasa93	CART	0	0	0	0	0	0	0	0
	QUICK	0	1	0	0	0	0	0	1
nasa93_center_1	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
nasa93_center_2	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
nasa93_center_5	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
sdr	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
albrecht	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
finnish	CART	0	0	0	0	0	0	0	0
	QUICK	1	1	1	1	1	1	1	7
kemerer	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
maxwell	CART	0	0	0	0	0	0	0	0
	QUICK	1	1	1	1	1	1	1	7
miyazaki94	CART	0	0	0	0	0	0	0	0
	QUICK	1	0	1	1	1	1	1	6

From Fig. 13, we see that 50th percentile corresponds to around a PRED(25) value of 2, which means that at the median point the performances of CART and QUICK are very close. Also note that 75th percentile corresponds to less than 15, meaning that for the cases when CART is better than QUICK, the difference is not dramatic. Our results show that the value added in using all the projects and features is limited. A QUICK analysis of just a small percentage of the data can yield estimates as good as the more complex learners like CART.

5.4 Detailed Statistical Analysis

Tables 5 and 6 compare QUICK to *passiveNN* and CART using seven evaluation criteria. *Smaller* values are *better* in these tables. When calculating "loss" for six of the measures, "loss" means higher error values. On the other hand, for PRED(25), "loss" means lower values. The last column of each table sums the losses of the method in the related row. If we sort the last column of Table 5, we see that the loss numbers are very similar:

$$\begin{aligned} QUICK: &\quad 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,6 \\ passiveNN: &\quad 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,3,6,6,7. \end{aligned}$$

The gray rows of Table 5 show the datasets where QUICK loses over most of the error measures (4 times out of 7 error measures, or more). The key observation here is that when using nearest neighbor methods, a QUICK analysis loses infrequently (only 1 gray row) compared to a full analysis of all projects.

As noted in Fig. 13, QUICK has a close performance to CART. This can also be seen in the number of losses summed in the last column of Table 6. The four gray rows in Table 6 show the datasets where QUICK loses most of the time (4 or more out of 7) to CART. In just $4/18 = 22\%$ of the datasets is a full CART analysis better than a QUICK partial analysis of a small subset of the data. The sorted last column in Table 6 is

$$QUICK: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 3, 6, 7, 7, 7$$
$$CART: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.$$

6 Sanity Check

As a sanity check of our results coming from publicly available datasets, we also ran our rig on more recent and proprietary data. Tukutuku is an active project which is maintained by Mendes [41] and the database contains 195 projects developed by 51 web companies from 10 different countries. Projects in Tukutuku are defined by 19 independent and one dependent variables (effort in man hours). We selected the dataset of the companies that have at least five or more projects, which yielded a total of 125 projects from eight different companies.

In proprietary data collection, there may be a tendency to define multiple categorical features [42]. However, some of the features may be uniform across many projects; hence, they do not help in differentiating projects from one another. The Tukutuku dataset is a good example of such a scenario. There is a variance difference between some features in orders of magnitude. A run of QUICK on the entire 19 features—disregarding the uniformity of the values in some features—resulted in a considerable performance decrease: QUICK was better than randNN in only two out of seven error measures.

A likely route to take in such cases is to use a feature selector such as linear discriminant analysis, wrapper, and so on [21]. However, understanding the pros and cons of different feature selectors, then choosing the appropriate one can be a deterring factor for a practitioner. Instead, we chose to use a very simple preprocessor of variance pruning, which includes removing features that are less than 25 percent of the median of all the feature variances. The variance-based feature removal process eliminated 8 of 19 independent features. After removing the eight uniform features, QUICK was statistically significantly better than randNN in all seven error measures.

QUICK's comparison to CART and passiveNN for eight different company datasets is given in Table 7. For convenience, the cases where QUICK loses for the majority of the error measures (4 or more) are highlighted. Note that there are only three highlighted rows for QUICK versus CART comparison, i.e., for five out of eight companies, QUICK is statistically significantly the same as CART. A similar scenario is also valid for QUICK versus passiveNN comparison, where QUICK is significantly the same as passiveNN for five out of eight companies. Hence, if a company is willing to invest resources 1) to collect data of hundreds of instances and tens of features and 2) to implement complex learners like CART, then there may be a performance gain, e.g., the three companies out of eight where QUICK is outperformed. However, as shown in Fig. 13, this gain may be marginal.

TABLE 7
QUICK's Sanity Check on Eight Company Datasets (Company Codes Are $C1, C2, \ldots, C8$) from Tukutuku

Dataset	Method	MMRE	MAR	Pred(25)	MdMRE	MBRE	MIBRE	MMER	SUM OF LOSSES
C1	CART	0	0	0	0	0	0	0	0
	QUICK	1	1	1	1	1	1	1	7
C2	CART	0	0	0	0	0	0	0	0
	QUICK	1	1	1	1	1	1	1	7
C3	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
C4	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
C5	CART	1	0	1	1	0	0	0	3
	QUICK	0	0	0	0	0	0	0	0
C6	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
C7	CART	0	0	0	0	0	0	0	0
	QUICK	1	1	1	1	1	1	1	7
C8	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
C1	passiveNN	0	0	1	0	0	0	0	1
	QUICK	1	1	0	1	1	1	0	5
C2	passiveNN	0	0	1	0	0	0	0	1
	QUICK	1	1	0	1	1	1	0	5
C3	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
C4	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
C5	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
C6	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
C7	passiveNN	0	0	0	0	0	0	0	0
	QUICK	1	1	1	1	1	1	1	7
C8	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0

Cases where QUICK loses for the majority of the error measures (4 or more) are highlighted.

Note that QUICK is an automated step of a broader process of *understanding* the data so as to come up with essential content. As we saw with the sanity check on the Tukutuku datasets, QUICK can benefit from expert judgment in certain cases where the experts manually look at the data. In our case, we were able to identify the variance-based filtering that removed uninformative columns. In the broader process of understanding the data, one should also consider the tradeoff between data collection effort and the cost of the filtering method.

7 Threats to Validity

A validity threat is the comparison of results against a baseline to make sure that observed effects are nonrandom. For that purpose, we compared the performances of passiveNN, CART, and QUICK against randNN (as seen in Fig. 12). We ran our standard check (which is used to produce Tables 5 and 6) and found out that randNN performs much worse than other methods. Since the comparison results are very uniform and uninformative, we only provided our comments.

Using data from a single source makes replication of research easier. It also exposes datasets to a wider audience so that likely errors in the datasets can be exposed more quickly. However, it also bears certain problems for different user groups:

1. *Why collect data:* For most companies, putting together scattered information into a software effort dataset requires extra effort. Unless user groups in companies are given explicit outcomes for their effort, this difficulty is likely to pertain. See [43] for a sample solution.
2. *Data privacy:* Another concern for the companies is exposing their proprietary information. This issue calls for data anonymization techniques in SEE.
3. *Proprietary-right period:* For research labs, sharing data can mean giving up the exclusive right to publish results from that dataset. The solution to that problem can be to define a proprietary-right period.
4. *Dataset aging:* The age of the datasets is another concern that lacks consensus. From a practitioner's point of view, there is limited value in using old datasets as they may no longer represent current standards. There are two likely solutions to handle the aging issue: 1) don't use datasets older than a certain amount of time; 2) identify the projects from older datasets that are still relevant for the current context. The first option is somewhat inefficient as it disregards all the instances of a dataset and is in disagreement with the transfer learning literature [44], [45], [46], which states that it is possible to transfer data between different contexts and time frames. The second option can be facilitated by transfer learning methods that can identify which instances can be transferred to contemporary contexts, i.e., which projects are still relevant. To the best of our knowledge, the only work that has previously questioned transfer learning between different time frames is that of Lokan and Mendes [47], [48]. Lokan and Mendes [47] found by filtering chronological sets of instances that time frame divisions of instances did not affect prediction accuracy. In [48], they found that it is possible to suggest a window size of a time frame of past instances which can yield performance increase in estimation. They also note that the size of the window frame is dataset dependent. Building on the prior findings to provide evidence of knowledge transfer through time frames can address the ageing issue of SEE datasets to a certain extent.

Another validity issue that should be mentioned is the use of pairwise comparisons to compare a large number of methods. This is an issue that poses a validity threat to a large number of SEE studies, including our research. Mittas and Angelis [49] note the problem with such pairwise comparison-based tests in their recent study. They describe the issue of Type-I error inflation inherent in pairwise comparisons and provide an alternative technique to assess multiple SEE methods. We expect more future work in this promising direction of addressing the threats of the pairwise tests.

The use of experts in an active learning guidance system is another validity threat. The assumption regarding experts is that they are able to collect and provide the actual effort value of the training set upon a query from QUICK. In an actual environment, it is hardly the case that experts can perfectly collect the actual effort value; it would be another estimate which deviates a certain amount from the actual effort. Since QUICK would use these nonactual, expert-estimated effort values for estimation, QUICK's estimates would be erroneous by a certain fraction right from the start. An intuitive way to model this deviation is to use a probability distribution function, say a normal distribution with a mean of 5 and a variance of 3, i.e., $N(5,3)$. Assume (for the sake of an intuitive example) that QUICK asked the experts for the actual effort of a training instance, whose actual effort value is 47 man months and also assume that $N(5,3)$ (which models expert error) returned the value of $+4$ for that particular case. Then, the value returned to QUICK by the experts would be $47+4=51$ and QUICK would be $\frac{(51-47)*100}{47}=8\%$ erroneous right from the start. This toy example can be extended in any number of ways to come up with interesting future work.

Note that keeping only the features with a popularity of zero may mean loss of information in certain scenarios. As part of an in-progress study, we are questioning QUICK variants where we observe the sensitivity of discarding synonyms with varying numbers of popularity. Our initial results show that including more synonyms does not provide a significant performance increase. Hence, our intuition is that there may be scenarios where discarding synonyms with popularity of greater than zero would mean loss of information. On the other hand, we did not yet detect such a scenario.

A threat to the validity of the results presented in this study is the particular choice of the error measures. Although we make use of a large sample of error measures based on the prior SEE research, the majority of the error measures are based on the relative error, e.g., MMRE, MdMRE, and Pred(25) are calculated using MRE. Among the seven error measures, only MAR is based on the absolute error. Absolute error measures are gaining more attention from the research community, e.g., Shepperd and MacDonell's [50] recent work proposes a new evaluation criterion based on absolute error. Note that we compare QUICK to CART and passiveNN on the majority vote of the seven error measures employed in this study and the choice of error measures (absolute or relative) may have an impact on the results. Although the error measures mostly give the same decision (see high sum-of-loss values in Tables 5 and 6), there are also cases where there seems to be disagreement (e.g., sum-of-loss values of 3 or 4). Also, we are unable to observe a certain pattern where a success or failure according to a certain error measure would certainly indicate the same or opposite scenario for another error measure. Therefore, inclusion of more absolute error-based error measures into the analysis and investigating the relation between decisions of different error measures (particularly absolute versus relative error) could be a good future direction.

8 Discussion

An important point in this research is how our results apply to contemporary commercial software development environments. We provide a discussion on the *"age issue"* of SEE datasets in Section 7. Building on that discussion, we should note that our datasets serve two different purposes: 1) proof of concept and 2) applicability of results.

The older datasets (e.g., cocomo*, nasa*, desharnais*) serve purpose #1 more—proof of concept. Function points and COCOMO are standard methods that can be used to collect data of contemporary projects. However, one can hardly say that software projects collected with these methods decades ago are the perfect representatives of the contemporary development environments. A perfect case in this issue would be to use QUICK from scratch on proprietary data. A practitioner willing to get a hint of the results regarding such a perfect case should consider the datasets closer to his/her own domain.

8.1 Need for QUICK-Like Methods

To understand the industry needs for QUICK-like automated methods to assist in reducing the complexity of data interpretation, we interviewed a total of five practitioners. Each of these practitioners apply data mining to real-world software engineering data and has years of experience in collecting data as well as building estimation methods. Our interviews revolved around the question of whether the automated data reduction methods like QUICK could offer any value added.

A benefit of QUICK-like methods as identified by multiple interviewees is their ability to reduce the number of projects and features to be discussed in a client meeting or in a Delphi commission. Given the complexities of real-world data and given the finite time for meetings with clients or other experts, it is impossible to discuss all aspects of all parts of all the data. QUICK-like methods can assist experts by providing essential content to be discussed. A further benefit that was identified comes from the fact that the data collection cost of features differs. For example, it was noted that collecting the actual effort value (i.e., dependent variable information) is far more costly than collection of independent variables, e.g., analyst capability or lines of code information. QUICK can identify essential projects for which an effort value is to be collected. Also, the cost of getting accurate data is one of the biggest cost factors in the data collection activity, e.g., one of the interviewees supports this idea with an example of $1.5M spent by NASA in the period 1987 to 1990 to understand the historical records of all their software in support of the planning activities for the International Space Station. QUICK can reduce the associated cost by identifying the subset of projects for which data need to be collected, instead of collecting the data of all the projects.

8.2 Application

A final, yet important, point of discussion that we would like to raise is the application of QUICK in an industry environment, i.e., how an industry practitioner can use the proposed approach. An example application process would be as follows:

1. Identify dependent and independent variables.
2. Decide the importance and easiness to collect of the independent variables.
3. Start collecting important and easy-to-collect independent variables.
4. Run QUICK to decide which of these features to keep.
5. Run QUICK to find popular projects, i.e., which would be used by ABE0-1NN.
6. Collect dependent variables of only the projects marked in the previous step.

9 Future Work

An interesting future direction may be the investigation of QUICK's effect on the linear regression methods, i.e., how the coefficients, the fitting, and the prediction accuracy of a regression method would be affected by the features and the instances chosen by QUICK.

QUICK uses distances between features after normalization. However, there are other alternatives that can be used for selection of features. For example, promising future work would be to investigate the effect of using a correlation coefficient-based feature selector in the synonym pruning step of QUICK.

10 Conclusion

The goal of this research is to investigate the essential content of SEE datasets and make recommendations regarding which estimation methods (simple or complex) should be favored. We define the essential content as the number of $F' \subseteq F$ features and $N' \subseteq N$ instances required to hold the information of a dataset.

Our results show that the essential content of SEE datasets is surprisingly small. Even the most commonly studied datasets (e.g., the *nasa93* dataset which could be summarized by only 4 percent of its actual size) can be summarized by a small portion of their features and instances. We also see that such a reduction protects the estimation performance. The implications of our research is twofold:

1. SEE datasets can be reduced to small essential content and, fortunately, simple methods are still able to perform well on the essential content.
2. QUICK can help to identify the important features and instances.

One final note: It would be inappropriate for this study to stop further research on complex SEE methods. It is tempting to increase the complexity of learners (e.g., use of a number of learners from machine learning). However, research on complex methods needs to discuss the value added in using such learners. In this paper, we show that at least the SEE datasets used in our research have small essential content and the value added in using complex methods is limited.

Acknowledgments

The work was funded by US National Science Foundation grants CCF: 1017330 and CCF: 1017263. The research reported in this document/presentation was performed in connection with contract/instrument W911QX-10-C-0066 with the US Army Research Laboratory. The views and conclusions contained in this document/presentation are those of the authors and should not be interpreted as presenting the official policies or position, either expressed or implied, of the US Army Research Laboratory or the US Government unless so designated by other authorized documents. Citation of manufacturers or trade names does not constitute an official endorsement or approval of the

use thereof. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon. The authors would like to thank Brad Clark, Brendan Murphy, Jairus Hihn, and Ricardo Valerdi for their invaluable feedback and time.

REFERENCES

[1] M. Jorgensen, "A Review of Studies on Expert Estimation of Software Development Effort," *J. Systems and Software,* vol. 70, pp. 37-60, Feb. 2004.

[2] M. Jorgensen and T. Gruschke, "The Impact of Lessons-Learned Sessions on Effort Estimation and Uncertainty Assessments," *IEEE Trans. Software Eng.,* vol. 35, pp. 368-383, May/June 2009.

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

[4] T. Menzies, O. Jalali, J. Hihn, D. Baker, and K. Lum, "Stable Rankings for Different Effort Models," *Automated Software Eng.,* vol. 17, pp. 409-437, Dec. 2010.

[5] M. Jorgensen and M. Shepperd, "A Systematic Review of Software Development Cost Estimation Studies," *IEEE Trans. Software Eng.,* vol. 33, no. 1, pp. 33-53, Jan. 2007.

[6] J.W. Keung, "Theoretical Maximum Prediction Accuracy for Analogy-Based Software Cost Estimation," *Proc. 15th Asia-Pacific Software Eng. Conf.,* pp. 495-502, 2008.

[7] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees.* Wadsworth and Brooks, 1984.

[8] A. Corazza, S. Di Martino, F. Ferrucci, C. Gravino, F. Sarro, and E. Mendes, "How Effective Is Tabu Search to Configure Support Vector Regression for Effort Estimation?" *Proc. Sixth Int'l Conf. Predictive Models in Software Eng.,* p. 4, 2010.

[9] Y. Li, M. Xie, and T. Goh, "A Study of Project Selection and Feature Weighting for Analogy Based Software Cost Estimation," *J. Systems and Software,* vol. 82, no. 2, pp. 241-252, 2009.

[10] T. Menzies, Z. Chen, J. Hihn, and K. Lum, "Selecting Best Practices for Effort Estimation," *IEEE Trans. Software Eng.,* vol. 32, no. 11, pp. 883-895, Nov. 2006.

[11] E. Kocaguneli, T. Menzies, and J. Keung, "On the Value of Ensemble Effort Estimation," *IEEE Trans. Software Eng.,* vol. 38, no. 6, pp. 1403-1416, Nov./Dec. 2012.

[12] M. Shepperd, "It Doesn't Matter What You Do but Does Matter Who Does It!" *Proc. CREST Open Workshop,* Oct. 2011.

[13] S. Dasgupta, "Analysis of a Greedy Active Learning Strategy," *Proc. Neural Information Processing Systems,* vol. 17, 2005.

[14] M.-F. Balcan, A. Beygelzimer, and J. Langford, "Agnostic Active Learning," *Proc. 23rd Int'l Conf. Machine Learning,* pp. 65-72, 2006.

[15] B. Wallace, K. Small, C. Brodley, and T. Trikalinos, "Active Learning for Biomedical Citation Screening," *Proc. 16th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 173-182, 2010.

[16] J.F. Bowring, J.M. Rehg, and M.J. Harrold, "Active Learning for Automatic Classification of Software Behavior," *ACM SIGSOFT Software Eng. Notes,* vol. 29, pp. 195-205, July 2004.

[17] T. Xie and D. Notkin, "Mutually Enhancing Test Generation and Specification Inference," *Proc. Int'l Workshop Formal Approaches to Software Testing,* pp. 1100-1101, 2004.

[18] A. Hassan and T. Xie, "Software Intelligence: The Future of Mining Software Engineering Data," *Proc. FSE/SDP Workshop Future of Software Eng. Research,* pp. 161-166, 2010.

[19] C.L. Chang, "Finding Prototypes for Nearest Neighbor Classifiers," *IEEE Trans. Computers,* vol. 23, no. 11, pp. 1179-1185, Nov. 1974.

[20] E. Kocaguneli, T. Menzies, A. Bener, and J.W. Keung, "Exploiting the Essential Assumptions of Analogy-Based Effort Estimation," *IEEE Trans. Software Eng.,* vol. 38, no. 2, pp. 425-438, Mar./Apr. 2012.

[21] K. Lum, T. Menzies, and D. Baker, "2CEE, A Twenty First Century Effort Estimation Methodology," *Proc. Joint Conf. Int'l Soc. Parametric Analysts and Soc. Cost Estimating and Analysis,* pp. 12-14, 2008.

[22] L.C. Briand, K. El Emam, D. Surmann, I. Wieczorek, and K.D. Maxwell, "An Assessment and Comparison of Common Software Cost Estimation Modeling Techniques," *Proc. 21st Int'l Conf. Software Eng.,* pp. 313-322, 1999.

[23] A. Bakr, B. Turhan, and A. Bener, "A Comparative Study for Estimating Software Development Effort Intervals," *Software Quality J.,* vol. 19, pp. 537-552, 2011.

[24] E. Alpaydin, *Introduction to Machine Learning.* MIT Press, 2004.

[25] B.W. Boehm, *Software Engineering Economics.* Prentice-Hall, 1981.

[26] A. Bakir, E. Kocaguneli, A. Tosun, A. Bener, and B. Turhan, "Xiruxe: An Intelligent Fault Tracking Tool," *Proc. Int'l Conf. Artificial Intelligence and Pattern Recognition,* 2009.

[27] A. Albrecht and J. Gaffney, "Software Function, Source Lines of Code and Development Effort Prediction: A Software Science Validation," *IEEE Trans. Software Eng.,* vol. 9, no. 6, pp. 639-648, Nov. 1983.

[28] B. Kitchenham and K. Känsälä, "Inter-Item Correlations among Function Points," *Proc. 15th Int'l Conf. Software Eng.,* pp. 477-480, 1993.

[29] C.F. Kemerer, "An Empirical Validation of Software Cost Estimation Models," *Comm. ACM,* vol. 30, pp. 416-429, May 1987.

[30] K.D. Maxwell, *Applied Statistics for Software Managers.* Prentice-Hall, 2002.

[31] Y. Miyazaki, M. Terakado, K. Ozaki, and H. Nozaki, "Robust Regression for Developing Software Estimation Models," *J. Systems Software,* vol. 27, no. 1, pp. 3-16, 1994.

[32] F. Walkerden and R. Jeffery, "An Empirical Study of Analogy-Based Software Effort Estimation," *Empirical Software Eng.,* vol. 4, no. 2, pp. 135-158, 1999.

[33] K. Dejaeger, W. Verbeke, D. Martens, and B. Baesens, "Data Mining Techniques for Software Effort Estimation: A Comparative Study," *IEEE Trans. Software Eng.,* vol. 38, no. 2, pp. 375-397, Mar./Apr. 2012.

[34] M. Shepperd and G. Kadoda, "Comparing Software Prediction Techniques Using Simulation," *IEEE Trans. Software Eng.,* vol. 27, no. 11, pp. 1014-1022, Nov. 2001.

[35] E. Mendes, I. Watson, C. Triggs, N. Mosley, and S. Counsell, "A Comparative Study of Cost Estimation Models for Web Hypermedia Applications," *Empirical Software Eng.,* vol. 8, no. 2, pp. 163-196, 2003.

[36] G. Kadoda, M. Cartwright, and M. Shepperd, "On Configuring a Case-Based Reasoning Software Project Prediction System," *Proc. UK CBR Workshop,* pp. 1-10, 2000.

[37] J. Keung, E. Kocaguneli, and T. Menzies, "Finding Conclusion Stability for Selecting the Best Effort Predictor in Software Effort Estimation," *Automated Software Eng.,* pp. 1-25, 2012.

[38] M. Shepperd and C. Schofield, "Estimating Software Project Effort Using Analogies," *IEEE Trans. Software Eng.,* vol. 23, no. 11, pp. 736-743, Nov. 1997.

[39] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit, "A Simulation Study of the Model Evaluation Criterion MMRE," *IEEE Trans. Software Eng.,* vol. 29, no. 11, pp. 985-995, Nov. 2003.

[40] J.P.C. Kleijnen, "Sensitivity Analysis and Related Analyses: A Survey of Statistical Techniques," *J. Statistical Computation and Simulation,* vol. 57, nos. 1-4, pp. 111-142, 1997.

[41] E. Mendes, "Cost Estimation of Web Applications through Knowledge Elicitation," *Proc. Int'l Conf. Enterprise Information Systems,* pp. 315-329, 2012.

[42] E. Kocaguneli, A. Misirli, B. Caglayan, and A. Bener, "Experiences on Developer Participation and Effort Estimation," *Proc. 37th EUROMICRO Conf. Software Eng. and Advanced Applications,* pp. 419-422, 2011.

[43] T. Menzies, C. Bird, T. Zimmermann, W. Schulte, and E. Kocaguneli, "The Inductive Software Engineering Manifesto: Principles for Industrial Data Mining," *Proc. Int'l Workshop Machine Learning Technologies in Software Eng.,* 2011.

[44] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer Learning for Cross-Company Software Defect Prediction," *Information and Software Technology,* vol. 54, no. 3, pp. 248-256, 2012.

[45] S.J. Pan and Q. Yang, "A Survey on Transfer Learning," *IEEE Trans. Knowledge and Data Eng.,* vol. 22, no. 10, pp. 1345-1359, Oct. 2010.

[46] A. Arnold, R. Nallapati, and W. Cohen, "A Comparative Study of Methods for Transductive Transfer Learning," *Proc. Seventh IEEE Int'l Conf. Data Mining Workshops,* pp. 77-82, 2007.

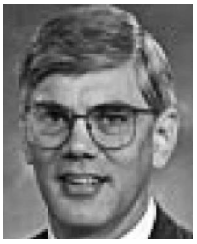
[47] C. Lokan and E. Mendes, "Using Chronological Splitting to Compare Cross- and Single-Company Effort Models: Further Investigation," *Proc. 32nd Australasian Conf. Computer Science,* pp. 47-54, 2009.

[48] C. Lokan and E. Mendes, "Applying Moving Windows to Software Effort Estimation," *Proc. Third Int'l Symp. Empirical Software Eng. and Measurement*, pp. 111-122, 2009.
[49] N. Mittas and L. Angelis, "Ranking and Clustering Software Cost Estimation Models through a Multiple Comparisons Algorithm," *IEEE Trans. Software Eng.*, vol. 39, no. 4, p. 537-551, Apr. 2013.
[50] M. Shepperd and S. MacDonell, "Evaluating Prediction Systems in Software Project Estimation," *Information and Software Technology*, vol. 54, pp. 820-827, 2012.

Ekrem Kocaguneli received the BSc and MSc degrees in computer engineering from Bogazici University. He received the PhD degree in computer science from West Virginia University. His main research interests are software estimation, artificial intelligence and machine learning applications in empirical software engineering, and intelligent tools to aid software processes. He is a student member of the IEEE.

Tim Menzies is a professor in computer science and electrical engineering at West Virginia University (WVU) and the author of more than 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 is the cofounder of the PROMISE conference series devoted to reproducible experiments in software engineering. In 2012, he was the cochair of the program committee for the IEEE ASE conference. He is a member of the IEEE.

Jacky Keung is an assistant professor in the Department of Computer Science at the City University of Hong Kong. Prior to joining CityU, 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. He is an active software engineering researcher, and his main research interests are in software cost estimation, empirical modeling and evaluation of complex systems, and data-intensive analysis for software engineering data and its application to project management. He is a member of the IEEE.



David Cok is a technical expert and leader in static analysis, software development, computational science, and digital imaging. He has been a major contributor to JML and Esc/Java2 for Java applications; his particular interests are the usability and application of static analysis and automated reasoning technology for industrial-scale software development. Prior to joining GrammaTech, he was a senior research associate in the Kodak Research Laboratory and held technical leadership and managerial positions. He is a member of the IEEE.

Ray Madachy is a research assistant professor in the Industrial and Systems Engineering Department and principal of the University of Southern California (USC) Center for Systems and SE. Prior to full-time research at USC, he was the chief science officer at Cost Xpert Group and a chief scientist at the C-bridge Institute. He is a coauthor of the book *Software Cost Estimation with COCOMO II* and is currently finishing the book *Software Process Dynamics*. His consulting clients have included Chevron, Northrop-Grumman, Seagate, Motorola, Lucent, USAA, Blue Cross, and other Fortune 500 firms. He is a member of the IEEE.