BETTER METHODS FOR CONFIGURING CASE-BASED REASONING SYSTEMS

by

Ekrem Kocagüneli

B.S, Computer Engineering, Boğaziçi University, 2008

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2009

ACKNOWLEDGEMENTS

Firstly I would like to present my heartfelt thanks to my supervisor Asist. Prof. Ayşe Bener for her support and guidance throughout my studies. Without her incredible energy while guiding me through my senior year as well as my master degree, I would not even come close to what I have achieved.

Secondly I am very grateful to Assoc. Prof. Tim Menzies, who was unbelievably generous while providing me with new ideas and spending his precious time for me throughout this research.

I would also like to thank Dr. A. Taylan Cemgil and Assoc. Prof. Necati Aras for their kindness to accept to be in my thesis jury. Many thanks to Dr. Jean Marc Desharnais as well as Dr. Jacky W. Keung for sharing their ideas and knowledge to shape my research.

I also owe special thanks to all SoftLab members, who provided for both the stimulating and fun research environment. I consider myself very lucky to get a chance to work together with them.

I am deeply thankful to my mother Zeliha, my father Ali and my brother Mehmet Ali for their unconditional love and support throughout my whole life and for always being there whenever I needed. The last but not the least, my thanks to Gizem Erdoğan for her support and her patience throughout my master studies.

Finally I would like to thank The Scientific and Technological Research Council of Turkey (TÜBİTAK) for supporting me with National Scholarship Program for MS students under the grant name BIDEB 2210. This research is also supported in part by TÜBİTAK under grant number EEEAG108E014, and by IBTECH A.Ş.

ABSTRACT

BETTER METHODS FOR CONFIGURING CASE-BASED REASONING SYSTEMS

Software effort estimation has been one of the major challenges in software engineering and previous research has mainly focused on addressing the large deviation problem in estimations by improving prediction accuracies of models. These models are evaluated using measures such as MRE or pred(r), which all assess the models on the basis of overall prediction accuracy.

Practitioners and researchers require a software effort estimation model with the following properties: 1)Understand the data that is used to build the model and 2) provide accurate estimations. In our study, we adapt greedy agglomerative clustering algorithm (GAC) to software effort estimation domain and use it as an analogy based estimator to build our model: **T**ree **E**stimation and **A**ssessment **K**nowledge (TEAK). By using GAC based model, TEAK, we are able to provide an analogy number (k) to be used for each individual test project and get lower MRE values than any other k-based method in all datasets.

There are multiple problems with case based reasoning (CBR) methods such as feature subset selection, scaling, similarity measure and number analogies to use (suitable k value) [1]. As our intention in this research was to focus on the problem of finding the suitable k value, we do not address other CBR related problems and stick to the dynamic selection of a suitable k value for each single test instance.

With TEAK it is possible to better understand the data on which effort estimation is to be done and use different number of analogies (k value) for each test instance. TEAK prunes irrelevant analogies in train set for a test project and thereby finds the

number of analogies to be used during estimation. This approach has outperformed all other k-based CBR methods in terms of predictive accuracy upto more than 100%.

ÖZET

DURUM TABANLI MUHAKEME SİSTEMLERİ İÇİN DAHA İYİ YAPILANDIRMA YÖNTEMLERİ

Yazılımda efor tahmini, yazılım mühendisliğindeki temel sorunlardan biri olagelmiştir ve daha önceki çalışmalar çoğunlukla modellerin tahmin kesinliğini arttırarak sonuçlardaki yüksek dalgalanmaları gidermeye odaklanmıştır. Bu modeller tümsel tahmin doğruluğuna dayanan MRE (görece hata büyüklüğü) veya Pred(r) gibi değerlendirme kriterleri kullanılarak değerlendirilir.

Uygulayıcılar ve araştırmacılar belli özellikteki yazılım efor tahmin modellerine ihtiyaç duyarlar. Bu özellikler: 1) Modeli kurmak için kullanılan veriyi anlayabilmek ve 2) kesin tahminler sunabilmek. Biz bu çalışmamızda açgözlü yığınlayıcı gruplama algoritmasını (AYG) yazılımda efor tahmini alanına uyarlıyoruz ve modelimizi (Ağaç Tahmin ve Değerlendirme Bilgisi - ATDB) kurmak için onu benzerlik tabanlı tahminleyici olarak kullanıyoruz. AYG tabanlı modelimizi, ATDB, kullanarak her bir test projesi için bir benzerlik sayısı (k) sağlamayı ve tüm veri kümelerinde diğer k-tabanlı yöntemlerden daha düşük MRE değerleri elde etmeyi başardık.

Durum Tabanlı Muhakeme (DTM) yöntemleriyle ilgili olarak özellik altkümesi seçme, ölçeklendirme, kullanılacak benzerlik ölçüsü ve kullanılacak benzerleri (uygun k değeri) seçme gibi pek çok sorun bulunmaktadir [1]. Bizim çalışmamızdaki temel amacımız uygun k-değeri bulmak olduğundan, DTM yöntemleriyle ilgili diğer sorunlar bu çalışmamızda irdelenmeyecektir.

ATDB ile veriyi daha iyi anlamak ve her test örneği için farklı sayıda benzerler

seçmek mümkün olmuştur. ATDB, test projesi için alakasız benzerleri budayarak kullanılacak uygun sayıdaki benzerleri bulur. Bu yaklaşım tahminsel kesinlik bakımından tüm diğer k-tabanlı DTM yöntemlerinden 100% ve üzerinde daya iyi sonuçlar vermiştir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

CBR	Case Based Reasoning
ABE	Analogy Based Estimation
FP	Function Points
k-NN	k Nearest Neighbour
ML	Machine Learning
kLOC	1000 Lines of Code
LOC	Line of Code
ISBSG	International Software Benchmarking Standards Group
SDR	SoftLab Data Repository

1. INTRODUCTION

One of the key challenges in software industry is the accurate estimation of the development effort. Accurate estimation of development effort is particularly important for risk evaluation, resource scheduling as well as progress monitoring [1]. Inaccuracies in estimations lead to problematic results, for instance overestimation causes waste of resources, whereas underestimation results in approval of projects that will exceed their planned budgets [2].

Although a significant number of methodologies have been proposed for effort estimation over the years, they have suffered from common problems such as very large performance deviations [3] as well as being highly dataset dependent. Comparative studies regarding best practices have therefore shown contradictory results [4].

In the recent years significant research effort was put into utilizing various machine learning (ML) algorithms as a complementary or as a replacement to previous methods [1] [5] [6] [7]. However, ML methods have an extremely large space of configuration possibilities [1]. When we consider configuration possibilities of ML methods induced on different datasets each having a characteristic of its own, it is not a surprise to see contradictory results [4] [1].

The predictive performance of any method is dataset dependent. Previous research has reported case based reasoning(CBR) or estimation by analogy [1] being able to produce more successful results in comparison to traditional regression based methods [7] [8]. Furthermore CBR has been favored over other methods when the dataset contains discontinuities [1]. However, it was also remarked that CBR techniques are subject to a variety of decisions that have a strong impact on its predictive performance. Such decisions include selection of features and/or instances, deciding on the number of analogies to be used and the adaptation strategy [1].

1.1. Motivation

In this research, we focus on deciding the number of analogies to be used (i.e. k nearest neighbours or projects [1]). Our claim is that we can avoid sticking to a fixed best performing number of analogies that changes from dataset to dataset. We use an alternative method to tune CBR techniques by proposing greedy agglomerative clustering algorithm (GAC).

GAC starts with the single project instances within the train dataset and treats them as the leaves of a GAC tree. Then it iteratively combines closest two projects to form a parent node, the same procedure applies for the parent nodes to build their parents and so on. At the end we end up with a binary tree build by GAC, whose leaves are the actual project instances and whose nodes are clusters of those single projects.

In our research, we form two GAC trees from the project data within our train set to build our model TEAK (**T**ree **E**stimation and **A**ssessment **K**nowledge). The first GAC tree (GAC1) is built on all the available train data and is used to decide on the node that has the lowest performance-variance, thereby pruning irrelevant train instances that would otherwise introduce higher variance and lower prediction accuracies on the results. The train instances which were clustered in the lowest performance-variance node are then used for building up the second GAC tree (GAC2). At the end our estimation becomes the median of the projects that are clustered in the lowest performance-node of GAC2.

Our aim by utilizing two GAC trees is to introduce a CBR calibrating method that makes its estimations via 1) building itself by discovering the characteristics of a particular dataset on its own and 2) pruning irrelevant instances on the basis of performance-variance. Our rigorous experiments that were conducted on 11 highly dissimilar software effort datasets have shown that GAC2 estimations are more than 100% less error prone than any fixed-k analogy approach (k is the number of analogies) on the basis of magnitude of relative error (MRE) subject to 95% Wilcoxon signed rank

test.

1.2. Outline

Up to this part of this thesis, a general introduction to software effort estimation studies is given. Furthermore we have also presented a general overview of CBR based methods and the motivation laying behind our research.

Section 2 provides information regarding related work. This research contributes to software effort prediction domain. Therefore, Section 2 provides background of the software effort estimation studies in general. Furthermore, our main aim is to promote research activities that would focus on understanding the characteristics of software effort datasets. For doing that we are using a GAC based method, which would be classified under calibration of CBR methods. For that reason, we will also provide background information concerning CBR methods in greater detail. Within the scope of this research, we have also formed a new software effort dataset. Formation of software effort dataset is more related to software measurement research than software effort. Although we gave brief information about software effort dataset formation brief, we wanted to provide some background information about that topic too, since it took about 3 months to form the dataset. We also think that some background information as well as our hands on experiences may be beneficial for other researchers too.

Section 3 provides the problem formulation. This section also includes the research questions that guided us throughout this research.

In Section 4 we present our proposed solution.

Section 5 defines the methodology we use to address research questions we raised, discusses the experimental settings, briefly summarizes the datasets we used and also provides descriptions of the performance measures under which we evaluated our proposed model. We also discuss the threats to validity of our research in this section.

Section 6 presents the results we obtained on 11 datasets. The results are evaluated under the performance measures that are defined in Section 5.

Lastly Section 7 summarizes our conclusions as well as the novel contributions of this research. In Section 7 we also suggest future directions of this research.

2. BACKGROUND

2.1. Overview

In this research we mainly focus on improving CBR techniques in software effort estimation domain. Therefore, we will be providing background information concerning both CBR as well as overall software effort estimation studies. In the context of this research, we have also formed a software effort estimation dataset, conducted a thorough analysis of each attribute and their relationships. Software measurement and dataset formation was not a direct objective of this research. However, understanding the principles of measurement, initiating a measurement process and collecting software effort data was a very labour intensive task, which took about 3 months to complete. Furthermore, there are a lot of pitfalls that are likely to be faced, whenever a measurement process is to be initiated and data are to be collected. Therefore, we will also give brief background information about software measurement in Section 2.4.

2.2. Software Effort Estimation

Software effort estimation has been extensively studied in literature since 70's. We can group effort estimation methods under two main categories: 1) Expert based methods and 2) Model based methods, where the former proposes making use of the experiences of human experts and the latter favours algorithmic approaches. The first model based methods were parametric models [9] [10] [11]. Although they generate successful results in certain datasets, parametric approaches suffered from local tuning problems when they were to be applied in another setting, i.e. they needed to be tuned to local data for high accuracy values [12]. On the other hand, local tuning requires collection of local data and the whole measurement and collection process is a challenge on its own [12] [3].

To address the local tuning problems of regression based methods, machine learning (ML) algorithms have been proposed as they do not require local tuning. Basically

ML based models learn from the past projects' effort data to make predictions for the future projects [13] [7] [14] [15].

However, industry's practices diverge from promising highly complicated formal methods and the mental process of effort estimation for software project managers is closer to CBR methods [16]. Our hands on experiences in software effort estimation model building in industry settings also gave us similar conclusions. Furthermore, software effort datasets are characteristically noisy datasets and CBR methods are more capable of handling noisy datasets with discontinuities than regression based models [16].

2.3. Case Based Reasoning

CBR can be defined as predicting the effort of a project by making use of similar past projects (analogies). To find the analogies, a number of methods can be employed: Nearest neighbour algorithms like k-NN, expert guidance or goal based preference [7]. To predict the number of analogies (k) while making an estimation is the challenge with CBR methods, since the number k plays a very critical role in the success of CBR methods [1]. In literature there are a number of studies proposing different k values in different contexts. Babu et al. address the issue of selecting prototypes, which they define as a process of finding representative patterns from data [17]. In their study Babu et al. use static selection rules to determine the best neighbourhood around a test instance and such rules show uniformity for all test set. On the contrary, we propose a selection scheme that uses dynamic rules, which are tuned per each single test instance. A further approach proposed for prototype optimization for k-NN uses a neural-net-based method. [18]. Method of Huang et al. is capable of updating more than one prototype in case of a misclassification and constructs not only the prototypes but also feature weights during the selection and optimization process. Therefore, from a theoretical point of view there is much space for adding in more machinery into standart estimation methods. However, there is hardly any strong evidence that this is useful in software effort estimation. Moreover, studies that involve much machinery induced on limited and not publicly accessible datasets may reduce the reproducibility

of experiments as well as their external validity. In our research we are proposing a prototype selection algorithm that uses single data structure called GAC tree twice (for GAC1 and GAC2).

Some studies favour using a pre-determined number of analogies in software effort estimation studies [19] [20] [21]. Lipowezky et al. propose a policy that looks for only one prototype, which can be regarded as extreme when dealing with datasets as small as those in software effort estimation [19]. Furthermore, we would like to base our estimations on some sample set of past data, not only on one record, since only one record may be misleading in small and heterogeneous datasets. Kirsopp et al. on the other hand propose making predictions from the 2 nearest cases as it was found as the optimum value for the datasets of their study [20]. In a further study Kirsopp et al. have increased their accuracy values with case and feature subset selection strategies [21]. Leaving the feature selection to future work, with dynamic selection of instances we have attained higher accuracy values than those reported for static rules or methods. Another study focusing on instance selection as well as feature weighting in context of CBR is conducted by Li et al. [22]. In their study Li et al. perform rigorous trials on actual and artificial datasets and they observe effect of various k values. However, we believe that reflection on dataset before applying to different algorithms under multiple settings is of more significance. In this research we propose making use of greedy agglomerative clustering (GAC) to address this challenge.

2.4. Software Measurement

In effort estimation studies, limited number of available datasets is one of the biggest challenges that researchers as well as practitioners face. For pure research purposes, one can suffice with utilizing the publicly available datasets. However, for software effort estimation studies in industrial settings publicly available datasets may not fulfil the expectations. In that case, a measurement initiative for a productivity model is inevitable.

While trying to measure software effort, size, duration or cost, we need to make

sure that the collected data meets some criteria. As Fenton et al. have put it, these criteria can be viewed as the baseline for *"good data"* [23].

- Correctness: Collecting the data in accordance with the definition that was developed for that particular kind of data is the definition of *correctness*. Therefore, a previous step to assure correctness would be to make sound definitions for each type of data or metric.
- Accuracy: *Accuracy* deals with the difference between the measured and the actual value of data. A good example that Fenton et al. provide here is the fact that a digital watch would provide more accurate values (closer to actual time) than an analog watch [23].
- Precision: As the name suggests, *precision* concerns the number of decimal places in a measure. From our experiences in the data collection phase of this thesis, we can say that unless being very sure that one can measure the precision that is given in the data, there is no need to make it more precise and introduce noise into dataset.
- Consistency: *Consistency* makes sure that one measure does not change from device to device or person to person. Probably consistency is one of the most important criteria among all. The reason for its importance lies behind the fact that in software engineering some of the data that you can not measure directly are tried to be elicited via questionnaires. The answers to the questions in questionnaires can be interval based like an answer that can take an integer value from 1 to 5. In that case, if the interval definitions are not done properly, then a 3 given by 2 different people can refer to absolutely different values.
- Time Interval: Whenever data refers to a particular time or time period, then a time stamp should be associated with that data.
- Replicability: Study results as well as past project repositories should be kept so that a future study can replicate the data collection of a past study and the past study can form a baseline for the future one.

Apart from ensuring the so called "goodness" of our data, we need to identify the entities we intend to measure. In software engineering we can group those entities

under 3 main categories: Processes, products and resources [23]. Since the focus of this research is not software measurement, we will give short descriptions and a few sample metrics of those categories. However, as further reading for extensive descriptions and sample metrics for each category, one can refer to the book of Fenton and Pfleeger [23].

- Process Metrics: Intention of process metrics is to capture the software-related *activities*. For example number of requirement changes, effort and number of specification or coding faults are among process metrics.
- Product Metrics: The artifacts that are produced by process activities are regarded as product metrics. As examples for product metrics we can name modularity, reuse and algorithmic complexity.
- Resource Metrics: Entities that process activities require are basically resource metrics. Some of the resource metrics are size and communication level of teams as well as age of personnel.

All the datasets we use and the one we collected in this research meet the above criteria, i.e. attributes come from process, product and resource types.

3. PROBLEM STATEMENT

Effort estimation in software engineering domain has various problems such as large deviations in performance or being highly sensitive to datasets [3]. To explain these concepts a little further, we can say that large deviations in performance refer to different results that were reported by different researchers [4]. *"Being highly sensitive to datasets"* mean that the performance of models change from dataset to dataset. A certain model that was tried on a limited number of datasets and that produced low error rates does not necessarily have the same performance when we try it on a different dataset. Dataset characteristics is another problem in effort estimation [1], since they only have a limited number of instances (for instance our dataset sizes change from 24 to 93, which is in fact a fairly small dataset) and they have very large variances in terms of their reported dependent variables (effort values) [1].

To address the accuracy issues, complex algorithms have been widely used. However, although putting a lot of effort in algorithms, some studies are reported to be tested only on a limited number of datasets [24]. Furthermore comparative research on these complex models have shown contradictory results [4]. In that respect, CBR methods have been reported to better represent datasets with discontinuities [1] [7] [8] and to produce higher accuracy values.

Although generating high accuracy values, CBR methods come with their inherent problems. The problems related to CBR methods are [1]:

- Deciding on which features to use
- Measure for calculating similarity
- Scaling
- Deciding on the number of analogies to use

Although each of the afore-mentioned CBR related problems are important ones to tackle, in this research our focus will be on the last one, i.e. the number of analogies

to use. In a study conducted by Kadoda et al. [1], the impact of the number of analogies selected while making predictions were investigated on effort data and it was reported that the underlying distribution of dataset had a decisive role on that number. Furthermore, they also suggested that the datasets come with their inherent complex properties and it requires a lot of time and effort to adapt a CBR system to a particular dataset [1]. In other words, we believe that the key to successful software effort estimation is to understand the dataset and to propose methods that are capable of doing that.

In our research we would like to answer the following research questions.

i. How can we better understand the dataset and the underlying characteristics of the dataset?
ii. How can we ease the procedure of selecting the number of analogies to be used in effort prediction as well as to improve the performance of current CBR methods?

4. PROPOSED SOLUTION

We propose a model called **T**ree **E**stimation and **A**ssessment **K**nowledge (TEAK) on the basis of performance-variance. We compare our proposed model with other analogy based methods in terms of magnitude of relative error (MRE), prediction at level r (Pred(r)) and win-tie-loss values.

4.1. GAC Model

Agglomerative clustering is a commonly used clustering method in various fields including data mining [25], database systems [26] and bioinformatics [27]. The popularity of making use of agglomerative clustering is its ability to use arbitrary dissimilarity or distance functions [28], which also makes it an appealing choice of clustering method for software effort data as software effort datasets also exhibit very dissimilar characteristics.

Greedy agglomerative clustering (GAC) is a greedy algorithm that starts with a set of instances and builds up a binary clustering tree with those instances according to a distance function [28]. In our research we use Euclidean distance function while building up our GAC tree. The formula of Euclidean distance function is given in Equation 4.1, where x and y correspond to two different projects, and x_i and y_i correspond to their i^{th} features.

$$Distance = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{4.1}$$

At the beginning of GAC, we can think of each individual project as a cluster of one instance. GAC then calculates the Euclidean distance between every two instance

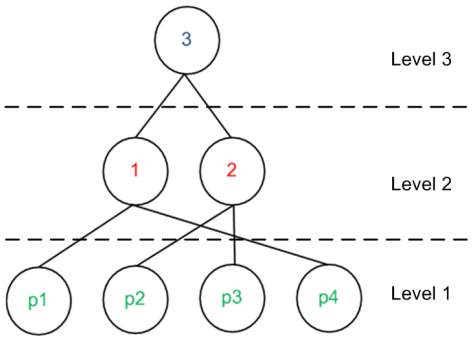


Figure 4.1. A Simple GAC Tree Example

and greedily clusters the closest two projects to form a cluster in one higher level of the tree. Then GAC continues to follow the same procedure at a higher level and so on until all the instances are clustered at a central cluster. Since the final structure is a binary tree as can be seen from Figure 4.1, we will use the terms cluster and node interchangibly from now on. Figure 4.1 is a very simple GAC tree that can be formed on a simple dataset of 4 projects. Let us assume that p1, p2, p3, and p4 are four initial projects of a hypothetical software effort dataset and they form Level 1 of the GAC tree. In that dataset p1-p4 and p2-p3 form the closes pairs of projects. Therefore, they are clustered to form the nodes at a higher level, Level 2. Finally node 1 and node 2 are clustered to form the node at Level 3 and since there are no more nodes to cluster, we are done building up the GAC tree.

In this research we are building up two GAC trees for our model, which are cascaded one after another. The pseudocode that describes how to build up a single GAC tree is given in Figure 4.2. In a given dataset of size n, the first GAC (GAC1) tree within TEAK is built upon n-1 instances. Once GAC1 is built, all its subtrees are traversed and their variances are stored. So as to remove the bias coming from numeric differences, stored subtree variances are scaled to $0-1$ interval. Then TEAK probabilistically selects subtrees according to their scaled variance values. The unique project instances that were within the selected subtrees are then used to form the second GAC tree (GAC2). Furthermore, TEAK enables its user to fine-tune GAC2 tree with the help of a variable called *bias* that represents the user's expert bias. The probabilistic selection of subtrees from GAC1 as well as how *bias* works can be seen from pseudocode given in Figure 4.3.

```
currentLevelNodes = Training Instances
levelIndex = 1
tree.level(levelIndex).nodes = currentLevelNodes
{As long as we did not get to root, continue forming GAC}
while size(currentLevelNodes) > 1 do
  nextLevelNodes = null
  levelIndex = levelIndex + 1
  repeat
    [closest1,closest2] = findClosest(currentLevelNodes)
    nodeToAdd = combineAsNewNode(closest1,closest2)
    tree.level(levelIndex).nodes.add(nodeToAdd)
    deleteFromCurrentLevelNodes(closest1,closest2)
  until size(currentLevelNodes) == 0
  currentLevelNodes = tree.level(levelIndex).nodes
end while
```

Figure 4.2. GAC Pseudocode

```
if normalizedVarianceNode_i ≤ rand()^bias then
  GAC2 ← GAC2 + Node_i
end if
```

Figure 4.3. Probabilistic Selection Pseudocode

We call our model, which is a special form of two cascaded GAC trees as **T**ree **E**stimation and **A**ssessment **K**nowledge (TEAK).

Finally once GAC1 and GAC2 are built, the estimation process starts. For estimation the test instance moves along the tree starting from the root until it finds the so called lowest-performance-variance node. The decision whether a test instance will move from one node to another node in the tree depends on comparison of variances of the current node and the weighted variance of its subtrees. Given three subtrees of a node containing $N = N1 + N2 + N3$ leaf nodes with variance values $V1, V2, V3$ then weighted variance beneath a node is calculated using Equation 4.2.

Dataset = *Cocomo81, Cocomo81e, Cocomo81o, Nasa93, Nasa93c2, Nasa93c5, Desharnais, Albrecht, SDR, SDR-Banking, ISBSG-Banking*
for $i = 1$ to 20 **do**
  $Dataset = randomize(Dataset)$
  for all T in Dataset **do**
    $testSet = T$
    $trainSet = Dataset$ minus T
    $GAC1$ = build GAC tree on $trainSet$
    $selectedSubtrees$ = start from root and probabilistically select subtrees of $GAC1$
    $GAC2$ = build GAC tree on unique instances of $selectedSubtrees$
    $finalNode$ = start from root and select lowest variance node from $GAC2$
    predicted effort for $T = median(finalNode)$
  end for
end for

Figure 4.4. TEAK Pseudocode

$$WeightedVariance = \frac{V1 * N1}{N} + \frac{V2 * N2}{N} + \frac{V3 * N3}{N} \tag{4.2}$$

If weighted variance of subtrees beneath a node is smaller than that of the node, then the test instance chooses to move to either left or right child that has the the lowest variance. On the other hand, if the weighted varince of subtrees is larger than that of their root node, test instance stays at that node. Finally the median of the train instances that were used to form the node at which test instance stops becomes the estimated effort value for that test instance, i.e. we choose to use k-many analogies for estimation where k is the number of train instances that are located in the selected node. The node at which the test instance stops is called the lowest-performance-variance node.

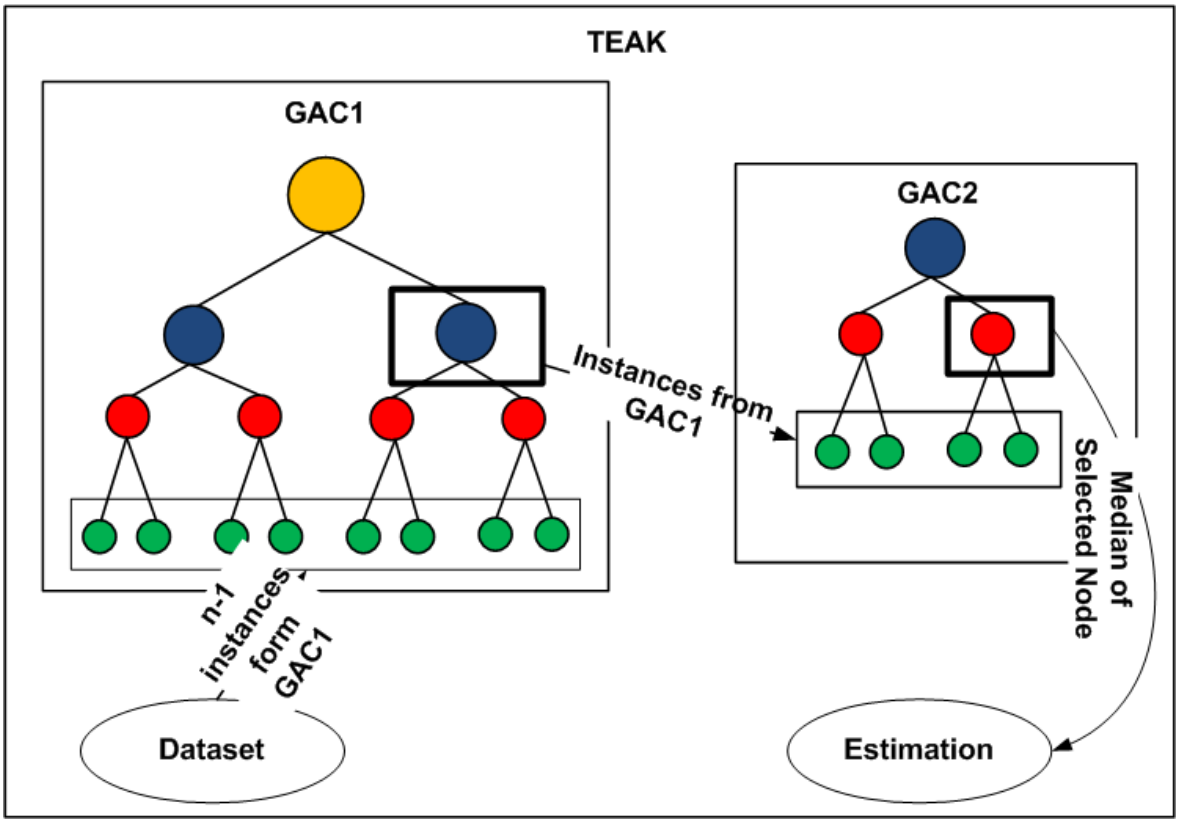


Figure 4.5. Execution of TEAK

The aforementioned estimation procedure is performed for each individual instance in an effort dataset separately. In other words, each instance within the dataset becomes a test instance for once and the remaining instances are used as its train instances to build the two GAC trees in TEAK. To better understand the TEAK algorithm, we provide the pseudocode in Figure 4.4 and we also provide a schema of the algorithm in Figure 4.5.

Our proposed model, TEAK, may use more than two GAC trees, since the dataset itself usually defines the number of GAC trees to be used. As long as we have enough instances to build a third or fourth GAC tree, there is no number limit for the GAC trees. However, since the size of our 11 datasets in this study range from 24 instances to 93 instances, the instances left for building up GAC2 can be as low as 2. Therefore, due to characteristics of our effort datasets, TEAK uses 2 GAC trees. Tricky point here is that unlike k-based CBR methods, TEAK does not need any expert interference to discover dataset characteristics so as to decide on the number of trees to be built or the number of analogies to be used in estimation. It decides on these numbers on its

own.

Regular k-based CBR methods start with a single analogy and increase this number depending on the overall performance of the whole dataset. Finally it uses the k value that yields the overall best performance. However, a fixed k value yielding *overall* best performance, does not necessarily provide the best performance for individual projects. We provide further details about the difference between individual and overall best performing k values in Section 6. Unlike regular k-based CBR methods, TEAK starts with all the instances in the train set and gradually prunes irrelevant instances on the basis of performance-variance. Furthermore, the pruning of irrelevancies is not performed for overall dataset but for each individual test instance, thereby considering the individual performances of each instance.

In that respect TEAK may resemble Ada Boost algorithm [29]. Ada Boost is basically composed of cascaded learners. At each round of Ada Boost, training set for a learner is selected probabilistically from the examples on which the previous learner performed poorly. Therefore, the selection criteria becomes the score of performances. The decision making then is performed by combination of learners via voting. The first difference of TEAK from standard Ada Boost is that it defines the number of learners (GAC trees) it will use by discovering the properties of the dataset. This feature makes it a better choice for software effort domain as datasets have varying characteristics. Secondly, while still probabilistically selecting train instances for GAC2, TEAK uses performance-variance so as to select train instances for GAC2. Finally, TEAK does not use voting of all learners. It uses the prediction of the the last learner (GAC2) as the prediction of the model.

5. METHODOLOGY

5.1. Data

Dataset characteristics are very important to evaluate the performance of a model. Since many previous models are criticized for having been tried on single or a very limited number of datasets [4] [1], we conducted our experiments on 11 different datasets among which 1 was formed within the scope of this research. 11 datasets used in this research come from 3 different sources.

Experiment replication is an important part of experimentation in software engineering as in any field [30]. Therefore, whenever necessary, we try to give as much details as possible regarding our work such as the details of algorithms and the used datasets. Within the following sections, we will group the datasets with respect to the source from which they come and we describe the content and characteristics of these datasets in greater detail.

5.1.1. PROMISE Data Repository

The first source we have used is PROMISE data repository [31]. PROMISE data repository is an on-line publicly available data repository and it consists of datasets donated by various researchers around the world. The datasets we have elicited via PROMISE data repository are: Cocomo81 [9], Nasa93 [32], Desharnais [33] and Albrecht [34]. Among those datasets Cocomo81 and Nasa93 are collected according to COnstructive COst MOdel (COCOMO)specifications, which were proposed by Barry Boehm [9]. On the other hand Albrecht and Desharnais datasets were collected in terms of function point (FP) count.

5.1.1.1. Cocomo81 Dataset. Cocomo81 dataset consists of 63 NASA projects collected from different centres during from 80s and 90s. As the name suggests the project data

collection for Cocomo81 was performed with respect to COCOMO cost model, in which project effort is measured in calendar months that consist of 152 hours. COCOMO assumes that the software effort is related to software size that is measured in terms of lines of code (LOC). However, the dependence of effort on software size is assumed to be more than just a linear dependence [32]. To see the "more than linear" effect of software size on software effort we can take a look at Equation 5.1. In Equation 5.1, the constants a and b are domain specific constants, whereas EM_i are effort multipliers, whose abbreviations as well as explanations can be found in Table 5.1. Effort multipliers can have one of the six values: Very low, low, nominal, high, very high, extra high.

$$Effort = a * (kLOC^b) * (EM_1 * EM_2 * EM_3 * ... * EM_{15}) \tag{5.1}$$

Table 5.1. Effort Multipliers in COCOMO Cost Model

Abbreviation	**Explanation**
ACAP	Analysts capability
PCAP	Programmers capability
AEXP	Application experience
MODP	Modern programming practices
TOOL	Use of software tools
VEXP	Virtual machine experience
LEXP	Language experience
SCED	Schedule constraint
STOR	Main memory constraint
DATA	Data base size
TIME	Time constraint for cpu
TURN	Turnaround time
VIRT	Machine volatility
CPLX	Process complexity
RELY	Required software reliability

5.1.1.2. Cocomo81e and Cocomo81o Datasets. In our research we also wanted to see the performance of our proposed model TEAK on homogeneous datasets in terms of a particular property. Therefore we only selected homogeneous datasets that are as big as the smallest heterogeneous dataset in terms of instance number.

Cocomo81 dataset enables the researchers to classify projects in terms of three different development modes: Organic, semi detached and embedded [9]. Therefore we used development mode as our breakdown criteria in Cocomo81 and took two homogeneous subsets of Cocomo81: Cocomo81o and Cocomo81e. Cocomo81o includes organic projects and Cocomo81e includes embedded projects.

5.1.1.3. Nasa93, Nasa93c2 and Nasa93c5 Datasets. Nasa93 dataset is similar to Cocomo81 dataset, in the sense that it is also collected in accordance with COCOMO cost model. In other words, the attributes of projects within this dataset are exactly same to those in Cocomo81 dataset. The number 93 in the name of the dataset is due to the fact that there are 93 project instances within the dataset. The projects in Nasa93 dataset are flight or ground system software projects that were developed for NASA in 6 different development centres between 70s and 80s [32].

As we have mentioned previously, we are also interested in homogeneous subsets of actual datasets to see the performance of TEAK. For Nasa93 dataset our breakdown criteria for forming homogeneous subsets was the development center of projects. Projects in Nasa93 are developed in one of the six different development centers. We took two subsets of Nasa93 dataset on the basis of their development centres: Nasa93c2 and Nasa93c5. Nasa93c2 is composed of projects that were developed in center 2 whereas Nasa93c5 contains projects that were developed in center 5. The reason for us to choose particularly center 2 and center 5 was the number of projects that were developed in these centres. Only those centres had developed as many projects as the smallest heterogeneous dataset.

5.1.1.4. Desharnais Dataset. Desharnais dataset is composed of software projects from a Canadian software house [33]. Dataset contains 81 projects and the size measure for this dataset is function points (FP). The attributes other than the size measure are given in Table 5.2.

Table 5.2. Attributes for Desharnais Dataset

TeamExp	Team experience (measured in years)
ManagerExp	Manager experience (measured in years)
YearEnd	The year in which the project ended (categorical)
Length	Actual project schedule in months
Effort	Actual effort measured in person-hours
Transactions	Number of basic logical transactions
Entities	Number of entities in the systems data model
PointsNonAdjust	Transactions + Entities
Adjustment	Function point complexity adjustment factor
PointsAjust	Function points adjusted by the Adjustment factor

5.1.1.5. Albrecht Dataset. Albrecht dataset contains 24 projects that were developed by using third generation languages that include COBOL, DMS and PL1 [35]. The independent features of Albrecht dataset are: Input count, output count, query count, file count, function points and source lines of code. The independent variable, effort, is recorded in terms of hours.

5.1.2. SoftLab Data Repository

Another source of data we used is the Bogazici University Software Engineering Research Laboratory (SoftLab) repository [6]. One of the datasets we have taken from SoftLab is SDR and it contains projects of various software companies from Turkey. SDR dataset is collected according to COCOMOII cost model [10] and the features included within SDR dataset are given in Table 5.3. Another dataset from SoftLab, SDR-Banking, is the dataset that we have formed within the context of this research (SDR does not include SDR-Banking). SDR-Banking dataset is composed of

software project data of a multinational bank in Turkey. The dataset is composed of 25 software projects that were developed within this bank. The dataset has 20 features that describe the characteristics of the software projects and an effort attribute related to the project. The features that are listed for each project have been defined so as to meet the specific needs of the bank and also we made sure to include product, process and resource measures into the attributes while defining them. We summarize the definition of each attribute within SDR-Banking dataset in Table 5.4. Depending on the characteristics of each feature, some of them have ordinal values whereas others have real values. We present these features here with their short explanations so that they can help other researchers and practitioners in defining their own measures.

5.1.3. ISBSG Data Repository

The last source we have used is the International Software Benchmarking Standards Group (ISBSG). ISBSG is a non-profit organization and it maintains ISBSG dataset, which is a project management dataset consisting of contributions from various companies and organizations [36].

Projects in ISBSG dataset can be grouped according to their business domains. In previous studies, breakdown of ISBSG according to business domain has also been used [37]. Among different business domains we selected banking due to:

i. Banking domain includes many projects whose data quality is reported to be high (ISBSG contains projects with missing attribute values).
ii. ISBSG Banking domain is the dataset we have analyzed and worked for a long time due to our hands on experience in building effort estimation models in banking industry.

We will represent the banking domain subset of ISBSG as ISBSG-Banking.

We provide further details regarding 11 datasets that we used in our research in two categories: 1) General properties and 2) statistical properties. General properties

include name of the dataset which we use throughout the paper, number of features and the number of instances within the dataset, whereas statistical properties include statistical facts concerning the independent variable (effort) such as mean, median, minimum, maximum and skewness. General properties of datasets can be found in Table 5.5 and statistical properties are reported in Table 5.6.

As we can see from Table 5.6, all the datasets have positive skewness and the skewness values range from 1.19 to 4.36, which indicates that the datasets are extremely heterogeneous with as much as 40-fold variation. We can conclude that the datasets diverge both in terms of their statistical characteristics as well as their domains and sizes. Therefore we make sure that we test the proposed model adequately.

5.2. Experimental Design

We used 11 different datasets from 3 different sources in our experiments to ensure that we address the criticism regarding previous studies in terms of trying models only on a limited number of datasets from limited sources and domains [24] [1] [4]. For each dataset we follow the same testing strategy, we use leave-one-out method [29] to select our test instance and use the remaining instances as our training instances to give to TEAK. We perform twenty runs on every dataset. Each run we select a random instance from the dataset, run TEAK on training instances and store our estimation, then we select another random instance and re-run TEAK. This estimation procedure is performed until all the instances within the dataset are used as a test instance.

For comparing TEAK with other k-based methods, we made use of various k values, since different researchers propose making use of different k values [1] [38]. Furthermore, number of analogies to be used for estimation, i.e. k-value is dataset dependent and plays a decisive role in estimation accuracy. Therefore, we included a wide range of k-values in our experiments so as to thoroughly compare the performance of TEAK to other k-based methods. In our experiments we used 5 static k values: 1, 2, 4, 8, 16. Also we exploited one dynamic k value, which is the best performing k value for each particular dataset. To find the best performing k value of a dataset, we

randomly selected ten instances from the dataset as our test set. Then we chose the k value that yielded the best performance for our test set as the best performing k value for that dataset.

5.3. Performance Measures

Our main performance measure that we use in our study is magnitude of relative error (MRE), which is basically the difference between the predicted and the actual value of a project effort value in terms of the actual value. Calculation of MRE for the i^{th} project in a dataset is given in Equation 5.2. Since we want to compare the estimation accuracy of our model to other models on a per instance basis, we used MRE as our comparison measure. We also used another common comparison criteria prediction at level r (Pred(r)) [37] and win-tie-loss values, which give performance of a model regarding the whole dataset. Therefore, our comparison criteria are capable of evaluating the performances of models both in terms of per instance basis as well as whole dataset performance.

$$MRE = \frac{|actual_i - predicted_i|}{actual_i} \tag{5.2}$$

As we have described in Section 5.2, we make 20 rounds for each dataset and in each round we use each individual project as a test instance. For 20 runs, we store the MRE values for both TEAK and other k values. After that we can compare performance of TEAK to any other k-based CBR method ($k = i, i \in 1, 2, 4, 8, 16, bestk$) on the basis of mean MRE values.

An MRE based performance measure that we use within this research is Pred(r). Pred(r) is the prediction at level r, i.e. the percentage of predictions whose error percentage is within $\pm$ r percentage of the actual value. The formula for calculating pred(r) is given in Equation 5.3.

$win_i = 0$, $tie_i = 0$, $loss_i = 0$

$win_j = 0$, $tie_j = 0$, $loss_j = 0$

if WILCOXON($MRE's_i$, $MRE's_i$) says they are different **then**

  $tie_i = tie_i + 1$;

else

  if mean($MRE's_i$) < mean($MRE's_j$) **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

Figure 5.1. Win-Tie-Loss Calculation Pseudocode

$$Pred(r) = \frac{100}{N} \times \sum_{i=1}^{N} \begin{cases} 1 \text{ if } MRE_i <= r \\ 0 \text{ otherwise} \end{cases} \tag{5.3}$$

In addition to MRE based methods, we also used win-tie-loss values to compare the performance of TEAK to other k-based CBR methods. While calculating win-tie-loss values, we first check if $method_i$ and $method_j$ are statistically different according to 95% Wilcoxon signed rank test in $round_k$. If they are not statistically different, then we increase tie_i and tie_j. On the other hand, if they turn out to be different, we check their mean MRE values for all test instances in $round_k$. Then the win value of method with lower mean MRE as well as the loss value of the method with higher mean MRE value is increased by one. The pseudocode for win-tie-loss calculation is given in Figure 5.1

Table 5.3. Features Defined for SDR Dataset

Feature Abbreviation	**Feature Definition**
PREC	Precedentedness
FLEX	Development Flexibility
RESL	Architecture and Risk Resolution
TEAM	Team Cohesion
PMAT	Process Maturity
ACAP	Analysts capability
PCAP	Programmers capability
AEXP	Application experience
MODP	Modern programming practices
TOOL	Use of software tools
VEXP	Virtual machine experience
LEXP	Language experience
SCED	Schedule constraint
STOR	Main memory constraint
DATA	Data base size
TIME	Time constraint for cpu
TURN	Turnaround time
VIRT	Machine volatility
CPLX	Process complexity
RELY	Required software reliability

Table 5.4. Features Defined for SDR-Banking Dataset

Feature Abbreviation	**Feature Definition**
PREC	Precedentness
TEAM	Team cohesion
PMET	Project methodology
ACAP	Analyst capability
RCAP	Architect capability
PCAP	Programmer capability
RECH	Overhead due to change in requirements
RDOM	Requirement team's domain knowledge
TMNO	Team number
SECU	Overhead due to security needs
THPT	Thirt party involvement
MODF	Module difficulty
COCO	Component count
MOCO	Module count
SERVICE	Service count
BATCH	Batch count
SCREEN	Screen count
REPORT	Report count
LOC	Lines of code
EFFORT	Effort in man hours

Table 5.5. General Properties of Utilized Datasets

Dataset	**# of Features**	**# of Projects**	**Content**
Cocomo81	17	63	NASA projects
Cocomo81e	17	28	Cocomo81 embedded projects
Cocomo81o	17	24	Cocomo81 organic projects
Nasa93	17	93	NASA projects
Nasa93c2	17	37	Nasa93 projects from center 2
Nasa93c5	17	40	Nasa93 projects from center 5
Desharnais	11	81	Canadian software projects
SDR	22	24	Turkish software projects
SDR-Banking	20	25	Banking software projects from Turkey
Albrecht	7	24	Projects from IBM
ISBSG-Banking	14	29	Banking projects of ISBSG

Table 5.6. Statistical Properties of Utilized Datasets

Dataset	**Mean**	**Median**	**Min**	**Max**	**Skewness**
Cocomo81	683.52	98.00	5.90	11400	4.36
Cocomo81e	1152.80	354.00	9.00	11400	3.37
Cocomo81o	59.87	46.00	6.00	240.00	1.68
Nasa93	624.41	252.00	8.40	8211.00	4.18
Nasa93c2	222.91	82.00	8.40	1350.00	2.37
Nasa93c5	1011.10	571.40	72.00	8211.00	3.46
Desharnais	5046.30	3647	546	23940	1.96
SDR	32.04	12.00	2	342	3.93
SDR-Banking	1779.40	1485	127	5727	1.19
Albrecht	21.87	11.45	0.5	105.20	2.15
ISBSG-Banking	5357.00	2355.00	662	36046	2.62

6. RESULTS

6.1. Results for Research Question 1

Our first research question was how we could better understand the underlying characteristics of dataset. Actually a majority of research studies in software effort estimation tries to address this question. However, as it was reported [24] [39] most of the methods in literature were tested on a single or a very limited number datasets, thereby reducing the credibility of the proposed method. To avoid this pitfall, we included many datasets from different domains and different sources. Although k-based CBR methods have been reported to produce better results than traditional regression based methods [40] [7] [8] and handle datasets with discontinuities better [24], they still have limited predictive power due to the fixed number of analogies that they use. In other words, a single best performing k value that is yielding the lowest MRE values for the whole dataset does not necessarily produce lowest MRE values for individual projects. We can better see this fact from Figures 6.1, 6.2, 6.3.

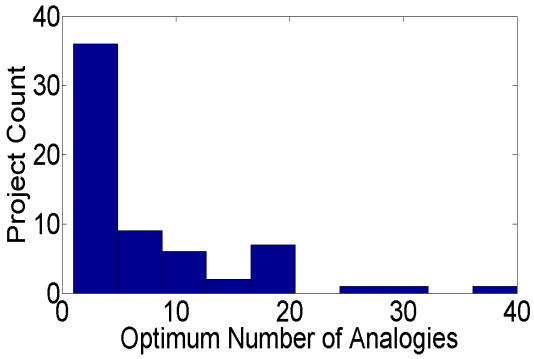


Figure 6.1. Distribution of Optimum k Values for Cocomo81

To produce histograms given in Figures 6.1, 6.2, 6.3, we calculated the optimum number of analogies, i.e. best-k value that yielded lowest MRE value for each project within a dataset. For a dataset of size n, optimum best-k value can range from 1 to $n-1$. What we can see from those figures is that for a considerable amount of projects a fixed-k approach will attain MRE values that are far from optimum. Since 3 datasets were enough to illustrate our viewpoint, we did not include all ten projects into this

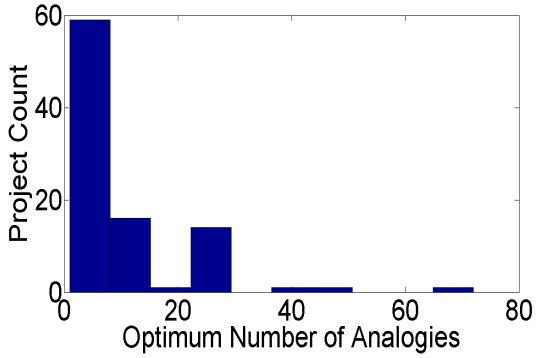


Figure 6.2. Distribution of Optimum k Values for Nasa93

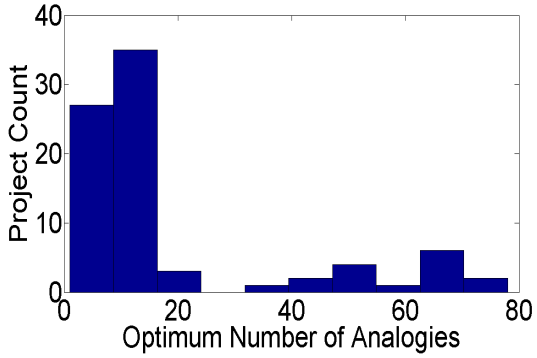


Figure 6.3. Distribution of Optimum k Values for Desharnais

analysis.

In this research we propose a model called TEAK that makes use of GAC trees. Unlike previous CBR calibration methods, TEAK does not use the overall performance of the whole dataset to decide on the number of analogies to be used. Also TEAK does not use a fixed number of analogies to be used. Instead TEAK prunes all unnecessary analogies on the basis of performance-variance for single projects rather than the whole dataset, thereby choosing the analogies to be used for each test instance dynamically. Furthermore, as we can see from Figure 6.4 and from related figures for each dataset, selecting small training sets on a per-instance basis making use of variance improves CBR effort estimation performance and outperforms fixed-k approaches. On the following subsections, we will present the MRE plots for each dataset that was used in the scope of this thesis and comment on those plots in terms of accuracies, which in return answers our first research question.

6.1.1. Experimental Results for Research Question 1

In this subsection, we will present the MRE plots, "win-tie-loss" values as well as Pred(25) values for each dataset separately. The details regarding the performance criteria such as "win-tie-loss" and Pred(25) were presented in Section 5.3.

The graphs presented for each dataset are basically sorted logged MRE values for each individual instance versus the instance number. On the x-axis, the instance numbers are given. Since we are using a leave-one-out approach during our experiments and since we are conducting 20 runs, the number of instances given on x-axis will basically be 20 times n, where n is the size of the dataset. The more important and informative part of the following graphs is their y-axis. On the y-axis, the MRE values for each test instance is given. However, to overcome the scalability issues while getting the plots, we applied a log filter to MRE values. Furthermore, to compare the performance of TEAK to other fixed-k approaches, we included all of them into the plots and sorted the MRE values associated with each approach.

Of course MRE is not the only approach for comparing our proposed model to others. We also use Pred(25) and "win-tie-loss" values to compare the performance of models with each other. While Pred(25) can be interpreted as another perspective of MRE calculation, "win-tie-loss" measures also take into account the significant difference of the results. While presenting the results, we paid particular attention to cover as many perspectives as possible. Yet more important than that we also evaluated our results on the basis of statistical significant tests, since a recent criticism that was put forward by Kitchenham et. al. was the fact that most of the software effort estimation studies lacked assessing their results in the light of statistical tests [24].

6.1.1.1. Results for Cocomo81 Dataset. We can see in Figure 6.4 that TEAK, which is represented with blue line achieves the lowest MRE values for majority of cases. On 6.4 blue line that represents the TEAK MRE values lay below all other methods. The closest to TEAK is the so called *"best K for dataset"* value. Almost on half of the test

instances -from 0 to around 550- all methods lay very close to each other and after the instance 550 the methods highly diverge from each other in terms of MRE values. Two points worth mentioning are: 1) In the first half, although the performances of MRE values lay very close to each other, TEAK has the lowest MRE values and 2) more important than the first remark, through the end of x-axis, most of the methods show very sharp increases in MRE values and TEAK is one of the three methods whose increase is a lot less than the other methods.

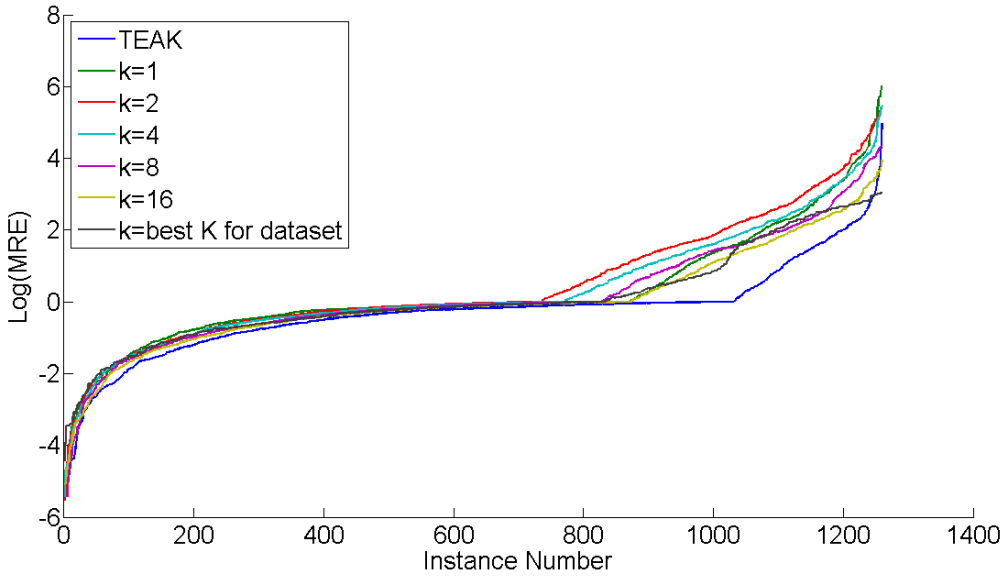


Figure 6.4. MRE Values for Cocomo81

Apart from the MRE plots, "win-tie-loss" values can be seen in Table 6.1. For Cocomo81 dataset, TEAK appears as yielding a win value of 108 which is more than twice that of its closest follower $k = bestK$ with a win value of 52. When we compare TEAK with other methods ($k = 1, 2, 4, 8, 16$), TEAK appears as a much more successful method, since it produces a "$win - loss$" value of 108, while "$win - loss$" values for all others turn out to be negative values. Furthermore, the *loss* value of TEAK is 0, which means that in significantly different cases TEAK had never been defeated by other methods in terms of overall MRE value. Therefore, for Cocomo81 win-tie-loss values confirm what we have seen in Figure 6.4, which is the fact that TEAK outperforms other methods significantly. The *Tie* value, which indicates how many times a model was not significantly different from other models, is also lowest for TEAK with a value of 132, which can be interpreted as the fact that TEAK produces not only lower MRE

values, but also they are significantly different than other methods most of the time.

Table 6.1. Cocomo81: Win-Tie-Loss Values for TEAK and multiple k values

Algorithm	**Win**	**Tie**	**Loss**	**Win - Loss**
TEAK	108	132	0	108
k=1	28	148	64	-36
k=2	16	136	88	-72
k=4	20	160	60	-40
k=8	34	170	36	-2
k=16	42	154	44	-2
k=bestK	52	180	8	44

6.1.1.2. Results for Cocomo81e Dataset. Both as previously mentioned in Section 5 and as the name suggests, Cocomo81e is a subset of Cocomo81 dataset. To mention it once more briefly here, Cocomo81 is a heterogeneous dataset which involves various software projects from various domains. Cocomo81e on the other hand is a homogeneous subset of Cocomo81, which only involves embedded system projects. The plot of Cocomo81e -Figure 6.5- is similar to that of Cocomo81, in the sense that again TEAK has the lowest MRE values in majority of the cases and again the jump in MRE values that is visible through the end of x-axis is the lowest for TEAK together with $k = 16$. What can also be drawn form Figure 6.5 is that the MRE lines are closer to one another than those in Figure 6.4. This situation can be attributed to the fact that the projects in Cocomo81e dataset are less in number and are similar to one another in comparison to those in Cocomo81 dataset.

As we see in Table 6.2, with a *win* value of 110, TEAK has significantly outperformed its closest follower whose win value is 72. For its other followers, we can say that TEAK has produced a *win* value that is 2 times to 18 times better than that of other k values. The *loss* value of TEAK for Cocomo81e dataset is 0, which means that TEAK always produced lower overall MRE values than any other method in significantly different cases. As a natural results of high *win* and $win - loss$ values, TEAK also has the lowest *tie* value.

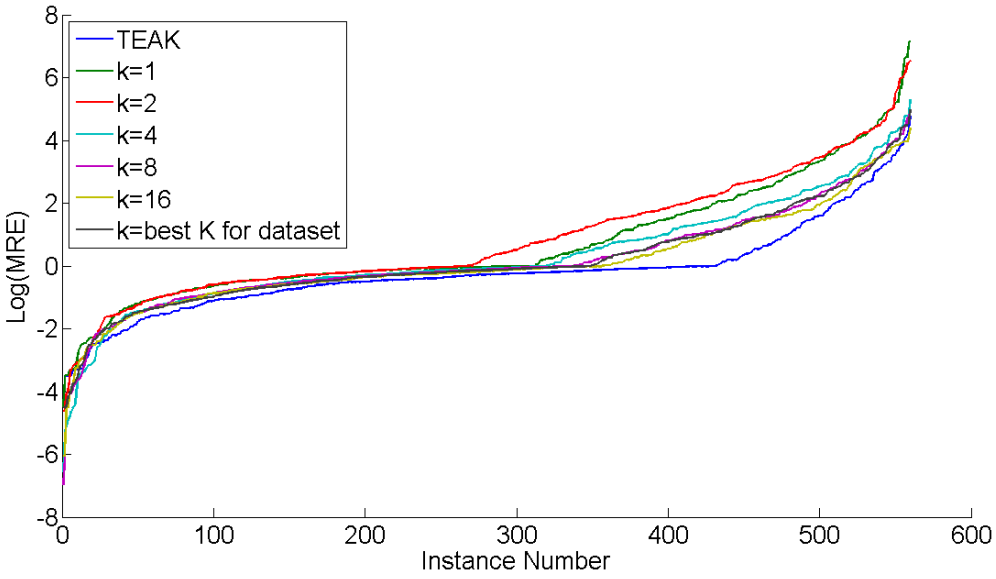


Figure 6.5. MRE Values for Cocomo81e

Table 6.2. Cocomo81e: Win-Tie-Loss Values for TEAK and multiple k values

Algorithm	**Win**	**Tie**	**Loss**	**Win - Loss**
TEAK	110	130	0	110
k=1	20	134	86	-66
k=2	6	118	116	-110
k=4	26	166	48	-22
k=8	34	168	38	-4
k=16	44	178	18	26
k=bestK	72	162	6	66

6.1.1.3. Results for Cocomo81o Dataset. Cocomo81o dataset is also a subset of Cocomo81 dataset. Like Cocomo81e, Cocomo81o is also composed of homogeneous project instances that were chosen from Cocomo81 dataset. The selection criteria for Cocomo81o is the type of projects, that is all the projects that form Cocomo81o dataset are organic type of projects. The definition for organic type of projects are given as the projects from relatively small software teams developing software in a highly familiar, in-house environment [9]. The performance of TEAK is better than all the other methods most of the time for that dataset as well. However, this time it competes with the MRE values yielded by $k = 16$ in some certain portions of the dataset, where the

blue line (for TEAK) lays above yellow line (for $k = 16$). This shows that for certain portions of datasets, although being a very small portion, TEAK can be challenged by other methods, which shows that those instances can further be identified and based on their characteristics TEAK may be improved. However, this will remain as a future work to another study as our main point here was to draw attention to the fact that we needed more effort on *understanding the data* and to propose a model that is more capable of doing that than any other previously suggested CBR method.

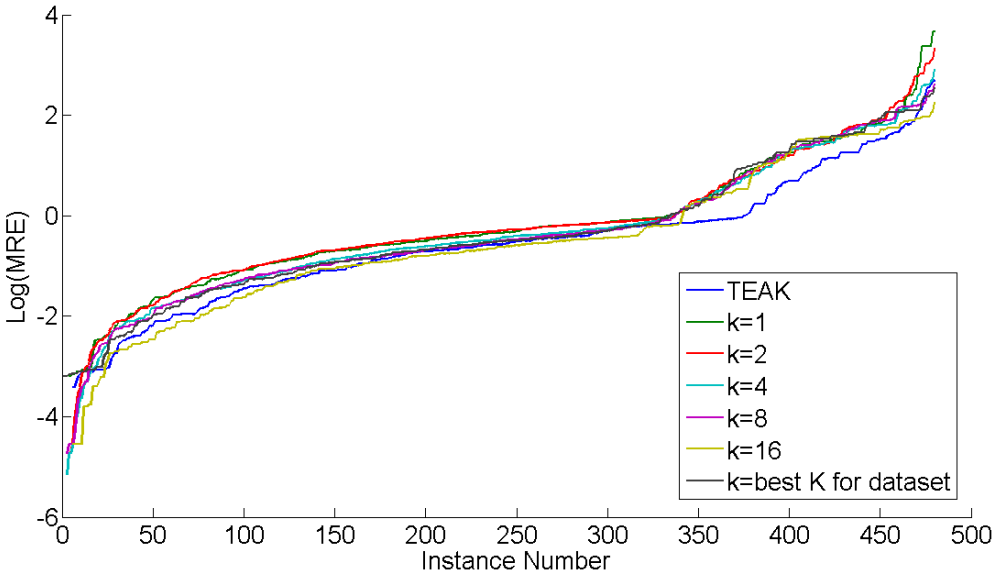


Figure 6.6. MRE Values for Cocomo81o

The win-tie-loss values in Table 6.3 indicates that TEAK was mostly challenged in Cocomo81o dataset up to now. As we see in Table 6.3, the Tie values of all methods are higher in comparison to those in Table 6.1 and in Table 6.2, which shows that Cocomo81o does not only challenge TEAK but it challenges all methods. Although being challenged by Cocomo81o dataset, TEAK still outperforms all other methods with a win value of 22. TEAK's win value is almost twice as much when compared with that of its closest follower, which is $k = 16$ with a win value of 12.

6.1.1.4. Results for Nasa93 Dataset. The general trend that we have observed in previous plots can also be seen in Figure 6.7, that is the lowest MRE values are attained by TEAK. Furthermore, another general trend that we have observed previously is the fact that as we go to the right of the x-axis, the interval between the lines or in other

Table 6.3. Cocomo81o: Win-Tie-Loss Values for TEAK and multiple k values

Algorithm	**Win**	**Tie**	**Loss**	**Win - Loss**
TEAK	22	218	0	22
k=1	2	196	42	-40
k=2	2	222	16	-14
k=4	8	228	4	4
k=8	8	232	0	8
k=16	12	228	0	12
k=bestK	8	232	0	8

words the gap between MRE values increases. What that means can be interpreted as the fact that most of the fixed-k approaches tend to make worse and worse estimations as the amount of MRE values increases. TEAK on the other hand is among the methods whose MRE increase is the lowest in comparison to others.

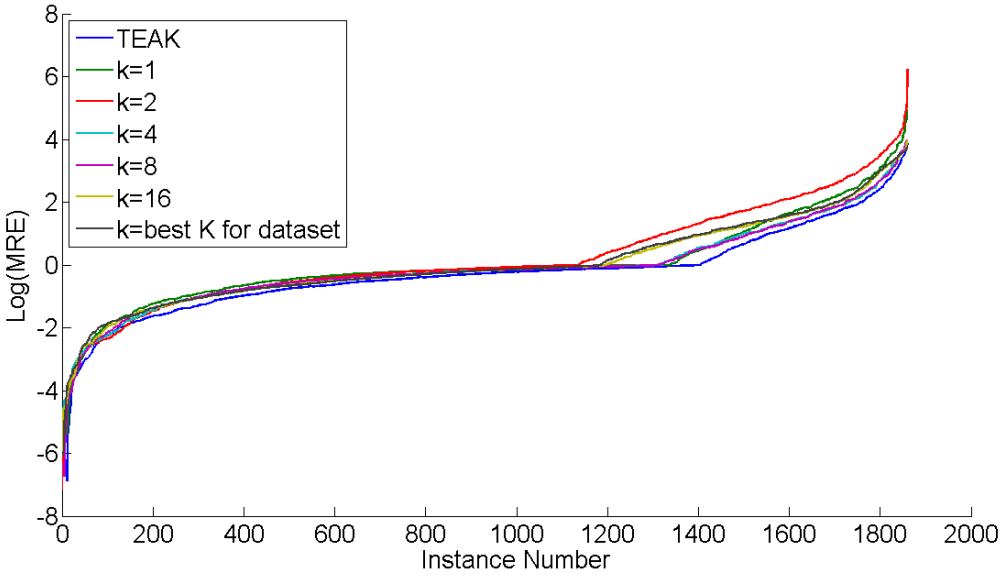


Figure 6.7. MRE Values for Nasa93

Nasa93 is another dataset that was collected in accordance with COCOMO cost model. In that respect Nasa93 is similar to Cocomo81 dataset. Therefore, we expect to see similar results in terms of TEAK's performance. Indeed, when we look at Table 6.4, TEAK outperforms other methods in terms of win-tie-loss values. For instance, in terms of *win* values, TEAK produces two times better results than the closest follower.

As for the other methods, TEAK produces up to 7 times better results in terms of *win* values. A difference from previous win-tie-loss values of other datasets is that TEAK is not the method with the lowest *tie* value in Nasa93 dataset. However, when we look at the $win - loss$ values in Table 6.4, we see that the methods that yielded the lowest *tie* were generating higher MRE values when being statistically different.

Table 6.4. Nasa93: Win-Tie-Loss Values for TEAK and multiple k values

Algorithm	**Win**	**Tie**	**Loss**	**Win - Loss**
TEAK	80	160	0	80
k=1	12	122	106	-94
k=2	18	132	90	-72
k=4	28	178	34	-6
k=8	40	182	18	22
k=16	38	194	8	30
k=bestK	44	192	4	40

6.1.1.5. Results for Nasa93c2 Dataset. In Figure 6.8, we can observe that all methods are very close to each other and up to the value 100 in x-axis, it is hard to tell which one outperforms the other. However, as we proceed through the x-axis, we see that TEAK outperforms all the other methods, not by a dramatic amount though. To better see how superior TEAK was to other k-based methods, we may refer to win-tie-loss values on Table 6.5.

As we can see from Table 6.5, Nasa93c2 dataset challenges all methods. We can see this fact from the low *win* and high *loss*, *tie* values. However, when we compare the performances of TEAK and other methods, we see in Table 6.5 that TEAK outperforms all other methods with the highest *win* as well as with the highest $win - loss$ values. Furthermore, again TEAK has the zero *loss* value, so TEAK has always outperformed other methods in terms of overall MRE, whenever its results were statistically significant.

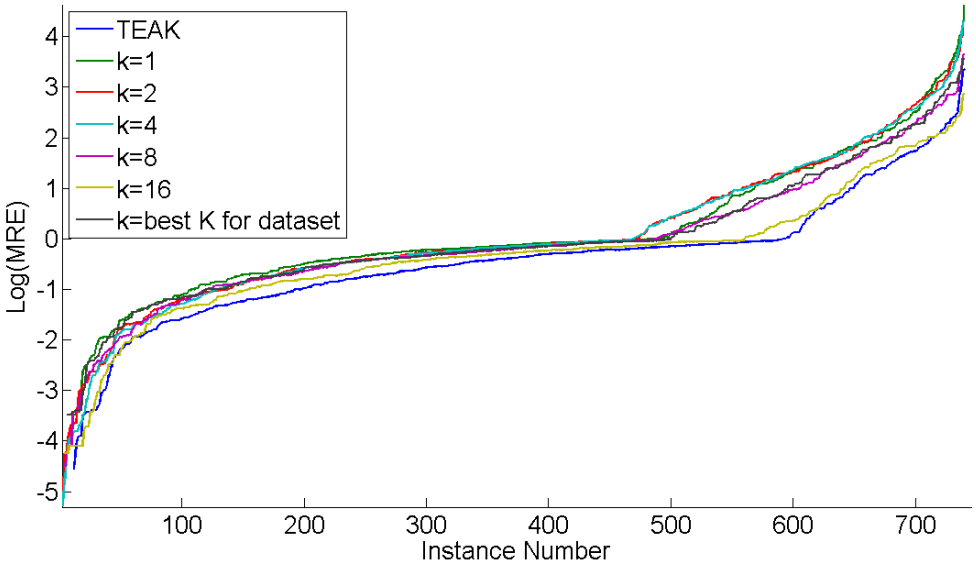


Figure 6.8. MRE Values for Nasa93c2

Table 6.5. Nasa93c2: Win-Tie-Loss Values for TEAK and multiple k values

Algorithm	**Win**	**Tie**	**Loss**	**Win - Loss**
TEAK	42	198	0	42
k=1	16	176	48	-32
k=2	2	166	72	-70
k=4	18	200	22	-4
k=8	28	208	4	24
k=16	26	206	8	18
k=bestK	28	206	6	22

6.1.1.6. Results for Nasa93c5 Dataset. The performance of TEAK in Nasa93c5 is similar to its performance in Nasa93c5. In other words, although being challenged by the dataset as all the other methods do, it still produces the lowest MRE values. We can see this fact from Figure 6.9.

The similarity of Nasa93c5 to Nasa93c2 that we observed in Figure 6.9 also reflects on the results in Table 6.6. We see that all methods have high *tie* values and low *win* values. To put it other words, results that methods produce are mostly not statistically different from one another. However, when they are statistically different, TEAK

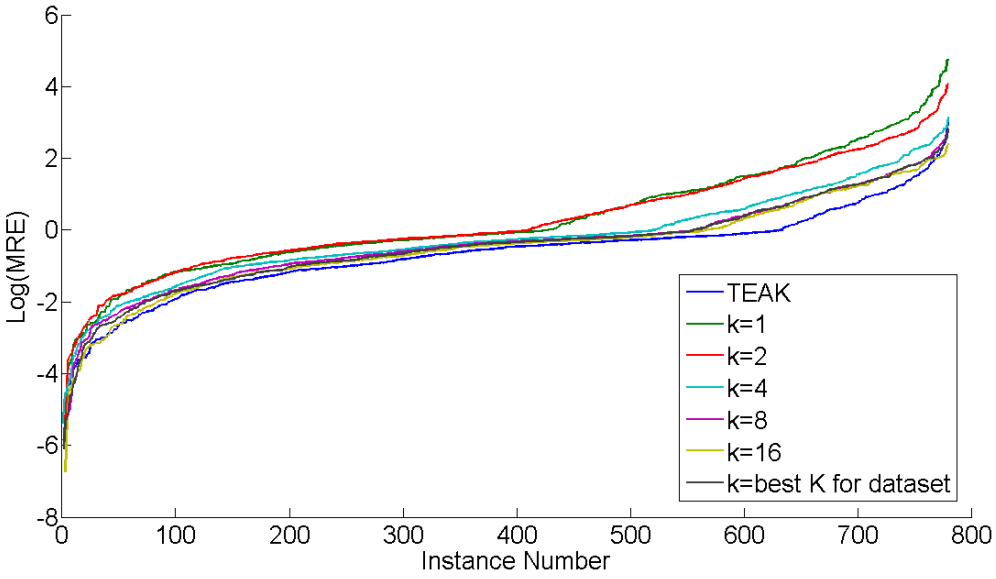


Figure 6.9. MRE Values for Nasa93c5

always produces lowest MRE values (*loss* value of zero). We see in Table 6.6 that TEAK's *win* value is the highest (48) and its *loss* value is zero.

Table 6.6. Nasa93c5: Win-Tie-Loss Values for TEAK and multiple k values

Algorithm	**Win**	**Tie**	**Loss**	**Win - Loss**
TEAK	48	192	0	48
k=1	2	172	66	-64
k=2	6	182	52	-46
k=4	12	200	28	-16
k=8	26	208	6	20
k=16	32	208	0	32
k=bestK	28	210	2	26

6.1.1.7. Results for Desharnais Dataset. We observe in Figure 6.10, that all k values attain very close MRE lines, which means that their performance is very close. In that case, we need to take a look at the win-tie-loss values so as to see whether the relatively small difference that we observe in Figure 6.10 have statistical meaning, i.e. whether these differences are statistically different. However, before consulting to win-tie-loss values, we can see from Figure 6.10 that for Desharnais dataset, TEAK is the model that attains the lowest MRE values as the blue line associated with TEAK lays below

the line of every other method.

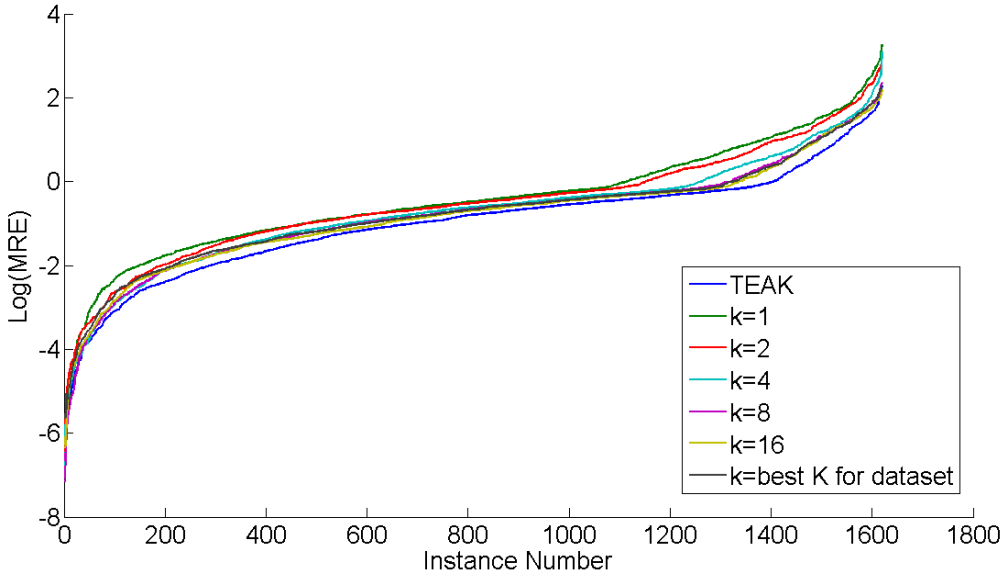


Figure 6.10. MRE Values for Desharnais

In Table 6.7, we see that TEAK produces *win* values that is almost 2 times better than its nearest follower and 32 times better than its worst follower, which indicates that the difference in lines that we observed in Figure 6.10 really have statistical significance. Furthermore, the statistical significance favours TEAK in the sense that both in terms of MRE plots and in terms of *win* values it comes out as the best performing method. Like the previous cases, again TEAK has a *loss* value of zero, which indicates that in none of the statistically significant cases has TEAK performed worse than any other method for Desharnais dataset.

Table 6.7. Desharnais: Win-Tie-Loss Values for TEAK and multiple k values

Algorithm	**Win**	**Tie**	**Loss**	**Win - Loss**
TEAK	64	176	0	64
k=1	2	140	98	-96
k=2	14	156	70	-56
k=4	32	192	16	16
k=8	30	202	8	22
k=16	34	200	6	28
k=bestK	30	202	8	22

6.1.1.8. Results for SDR Dataset. Probably among all plots, the plot given in Figure 6.11 for the MRE values of SDR dataset shows the greatest divergence between the performance of TEAK and other methods. What we see in Figure 6.11 is that when all the k-based methods perform very close to each other (very close lines), the blue line that represents TEAK lays a lot lower than all the other methods. For the line of TEAK to lie a lot lower than the lines of other methods is a strong indicator that it outperforms any other k-based method considerably. This situation is also a proof that TEAK is much more capable of understanding the dataset characteristics better than other k-based CBR methods.

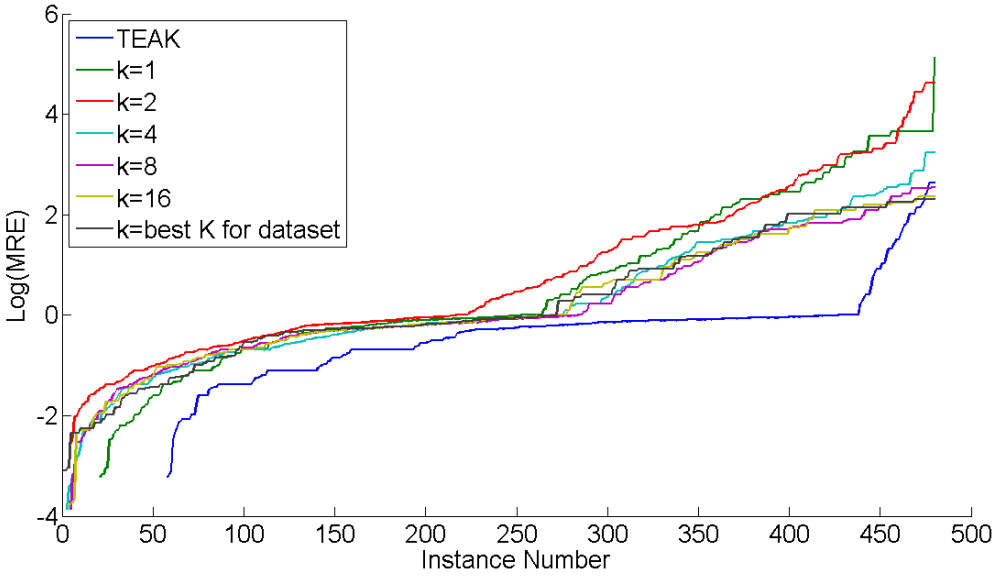


Figure 6.11. MRE Values for SDR

A statistical confirmation of the facts we have observed in Figure 6.11 is given in the table 6.8. We can see in Table 6.8 that TEAK outperforms other methods 4 times to 52 times in terms of *win* values. The *loss* value of zero also confirms that in none of the statistically significant cases has any other method beaten TEAK in terms of overall MRE values. Moreover, TEAK has the lowest *tie* value and the highest $win - loss$ value. Those figures also state that TEAK is performing much better than other methods in SDR dataset.

6.1.1.9. Results for SDR-Banking Dataset. As we have mentioned previously, SDR-Banking dataset has been formed in the context of this thesis. The performance of

Table 6.8. SDR: Win-Tie-Loss Values for TEAK and multiple k values

Algorithm	**Win**	**Tie**	**Loss**	**Win - Loss**
TEAK	102	138	0	102
k=1	20	176	44	-24
k=2	2	170	68	-66
k=4	16	202	22	-6
k=8	26	196	18	8
k=16	18	188	34	-16
k=bestK	22	198	20	2

TEAK on SDR-Banking dataset is important in the sense that it gives an idea of its performance on a dataset, which was formed particularly so as to meet the needs of a specific software company in a specific domain. We can see from Figure 6.12 that TEAK outperforms 3 other static k-based methods for SDR-Banking dataset up to instance number 350 and is being outperformed by the other 3. After instance number 350 it is difficult to claim that one method is superior to the others, since the lines lie very close to one another. Therefore, SDR-Banking dataset appears as one of the most challenging datasets both for TEAK and for other datasets as well.

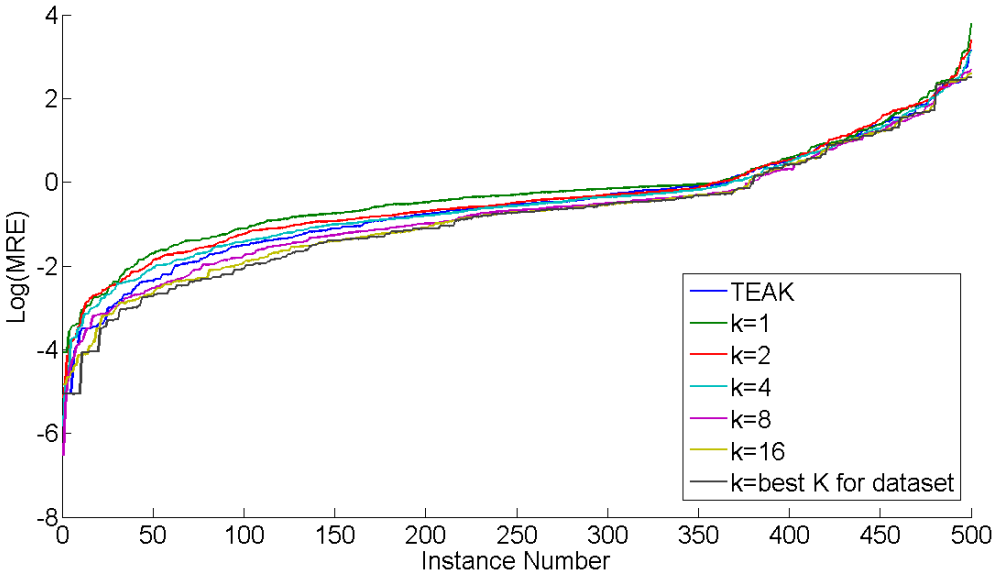


Figure 6.12. MRE Values for SDR-Banking

Another comparison criteria we used this research are win-tie-loss values which

are summarized in Table 6.9 for SDR-Banking dataset. As we can see from Table 6.9, TEAK does not have the highest *win* or the highest $win - loss$ value this time. This is due to the fact that SDR-Banking dataset is the most challenging dataset among all the tried datasets. Furthermore SDR-Banking was collected from the field for this research and the project instances within the dataset show a great diversity both in terms of their development methodologies, their development purposes as well as their sizes. However, the *loss* value of 0 shows us that similar to many other dataset results TEAK has never attained a mean MRE value that is worse than other methods in statistically significant cases.

Table 6.9. SDR-Banking: Win-Tie-Loss Values for TEAK and multiple k values

Algorithm	**Win**	**Tie**	**Loss**	**Win - Loss**
TEAK	12	228	0	12
k=1	0	160	80	-80
k=2	12	214	14	-2
k=4	14	212	14	0
k=8	20	220	0	20
k=16	24	216	0	24
k=bestK	26	214	0	26

6.1.1.10. Results for Albrecht Dataset. When we compare the MRE plots of other datasets to the plot of Albrecht dataset that is given in Figure 6.13, we see that Albrecht is one of the most challenging datasets. From Figure 6.13 it is hard to claim that TEAK or any other method is superior to the others. However, we can see that TEAK and $k = 16$ competes with each other, i.e. partially TEAK and partially $k = 16$ yields lowest MRE values. Again for the numerical and statistical check of Figure 6.13, we need to resort to win-tie-loss values, which are given in Table 6.13.

Table 6.13 does not actually tell anything that is much different than what we observed from Figure 6.13. In other words, win-tie-loss values given in Table 6.13 confirms statistically that Albrecht is the most challenging dataset for all k-based methods as well as for TEAK. When we take a look at the *win* values, we see that only

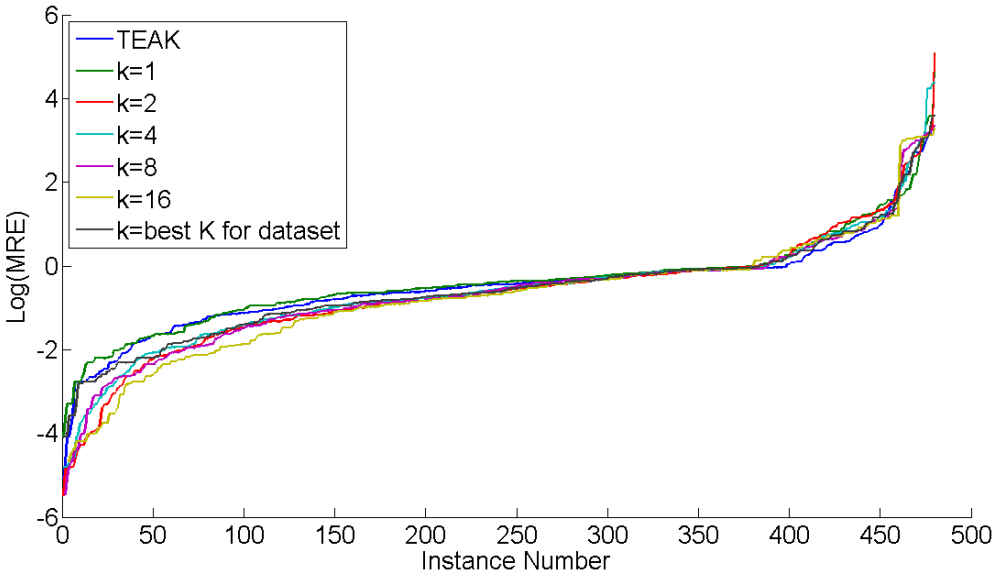


Figure 6.13. MRE Values for Albrecht

three methods can have positive *win* values, which are TEAK, $k = 8$ and $k = 16$. We see that all the three have very close *win* values, yet TEAK together with $k = 16$ has the best *win* value of 6. One further thing that takes attention almost immediately, when we try to interpret the figures of Table 6.10 is the fact that the *tie* values are extremely high. What high *tie* values tell us is that in most of the cases regardless of the high or low MRE values, the methods fail to gain statistical significance, i.e. although having different MRE values they are essentially the same.

Table 6.10. Albrecht: Win-Tie-Loss Values for TEAK and multiple k values

Algorithm	**Win**	**Tie**	**Loss**	**Win - Loss**
TEAK	6	234	0	6
k=1	0	230	10	-10
k=2	0	234	6	-6
k=4	0	240	0	0
k=8	4	236	0	4
k=16	6	234	0	6
k=bestK	0	240	0	0

6.1.1.11. Results for ISBSG-Banking Dataset. The ISBSG-Banking dataset can be regarded as slightly challenging when we take a look at the MRE plot given in Figure

6.14. In Figure 6.14, we see that between instance number 0-50 as well as between 250-300, TEAK is challenged by other methods. However, when we observe all instances which are close to 600 instances, we easily see that TEAK is more successful than other methods. One further point regarding ISBSG-Banking dataset is the fact that TEAK attains the lowest top MRE value, which we can observe from the right end of the plot given in Figure 6.14. Although TEAK is the most successful method in terms of MRE plots, we still need to check out the win-tie-loss values to be certain about this fact as the MRE lines lay very close to each other.

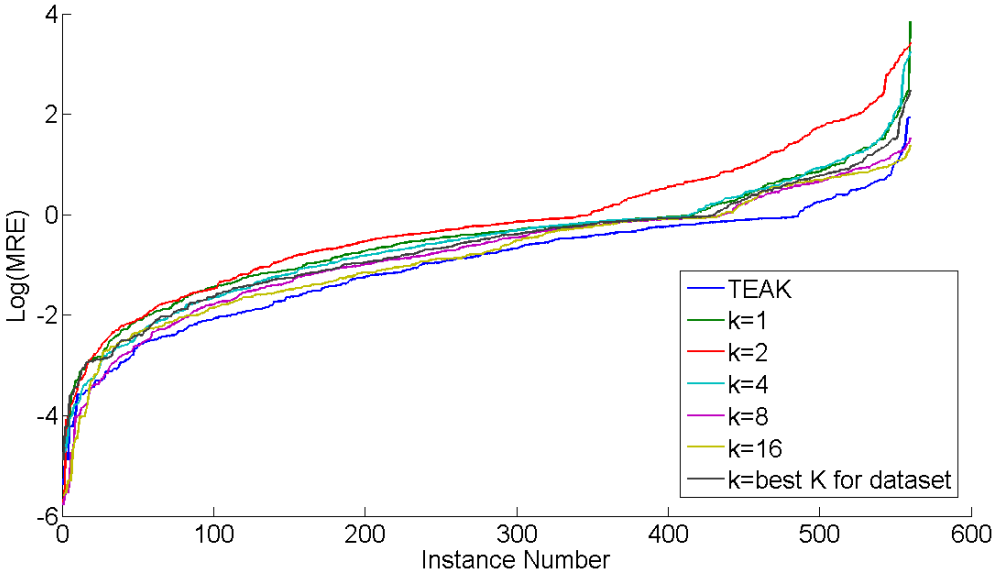


Figure 6.14. MRE Values for ISBSG-Banking

If we look at win-tie-loss values given in Table 6.11, we see that TEAK has the highest *win* value of 54 and a loss value of 0. Therefore, we can say that TEAK has produced lowest overall MRE values in most of the cases. Furthermore, a *loss* value of zero also confirms that no other method has produced a mean MRE value that is lower than TEAK in mutual comparisons, which turned out to be statistically significant.

6.1.1.12. Pred(25) Results for All Datasets. In the previous subsections, we have evaluated the performance of TEAK as well as other methods in terms of the evaluation criteria of MRE and win-tie-loss values. Although MRE and win-tie-loss values are fairly enough in terms of assessing the performance of models from different perspectives, we provide a further evaluation criteria of Pred(25), which is also a commonly

Table 6.11. ISBSG-Banking: Win-Tie-Loss Values for TEAK and multiple k values

Algorithm	**Win**	**Tie**	**Loss**	**Win - Loss**
TEAK	54	186	0	54
k=1	6	154	80	-74
k=2	8	154	78	-70
k=4	18	194	28	-10
k=8	36	196	8	28
k=16	44	196	0	44
k=bestK	34	200	6	28

used evaluation criterion in software effort estimation studies. To briefly remind the meaning of Pred(25), we can say that Pred(25) is the ratio of the instances that were estimated within 25% error range to the number of all the instances in a given test set. Therefore, Pred(25) can be interpreted as another perspective of MRE.

The mean and standard deviation of Pred(25) values for all datasets are given in Table 6.12 and Table 6.13 respectively. The reason why we provide mean and standard deviation instead of single Pred(25) value is due to the fact that we perform 20 runs for each dataset. Consequently for an algorithm induced on a single dataset, we have 20 Pred(25) values. Conducting 20 different runs instead of a single run greatly reduces the bias inherent in a particular dataset-algorithm configuration. Although we provided the standard deviation for the Pred(25) values in Table 6.13, we will base our comments on the mean Pred(25) values. The reason why we provided standard deviation results is only to give a sense of the spread of Pred(25) values that yielded the mean Pred(25) values.

As we can see from the mean Pred(25) values given in Table 6.12, TEAK always yields the highest Pred(25) values for all the datasets except Albrecht and SDR-Banking. As we could remember from Section 6.1.1.10, SDR-Banking and Albrecht were the most challenging datasets for TEAK as well as for other algorithms and we have concluded in the same Section that it was difficult to claim a best performing algorithm for these datasets. However, on all the other datasets, in terms of exten-

Table 6.12. All Datasets: Mean Pred(25) Values for TEAK and multiple k values

Dataset	**TEAK**	**k=1**	**k=2**	**k=4**	**k=8**	**k=16**	**k=best K**
Cocomo81	0.13	0.09	0.09	0.09	0.1	0.11	0.1
Cocomo81e	0.24	0.16	0.15	0.19	0.18	0.24	0.2
Cocomo81o	0.14	0.07	0.08	0.1	0.1	0.1	0.1
Nasa93	0.15	0.1	0.12	0.11	0.11	0.11	0.11
Nasa93c2	0.17	0.1	0.11	0.12	0.11	0.14	0.1
Nasa93c5	0.21	0.11	0.11	0.15	0.17	0.18	0.19
Desharnais	0.3	0.17	0.2	0.24	0.26	0.27	0.26
SDR	0.22	0.13	0.06	0.1	0.08	0.09	0.12
SDR-Banking	0.28	0.18	0.16	0.28	0.32	0.36	0.36
Albrecht	0.15	0.15	0.23	0.21	0.23	0.26	0.22
ISBSG-Banking	0.33	0.19	0.18	0.23	0.26	0.3	0.23

sive the measurement methods we have used in this research TEAK has significantly outperformed all other methods.

6.1.1.13. Pred(25) Results for All Datasets with Single and Complete Linkage. For agglomerative clustering algorithms single, average and complete linkages can be used as the distance measure for the construction phase [41]. The brief explanation of each distance measure is as follows:

- Complete-Link Clustering: Combine two cluster with smallest maximum pairwise distance.
- Average-Link Clustering: Combine two clusters with average of distances between all pairs in clusters.
- Single-Link Clustering: Combine two clusters with smallest minimum pairwise distance.

We have conducted our previous experiments by making use of average-link clustering. However, depending on the dataset, other distance methods such as complete-

Table 6.13. All Datasets: Standard Deviation of Pred(25) Values for TEAK and multiple k values

Dataset	**TEAK**	**k=1**	**k=2**	**k=4**	**k=8**	**k=16**	**k=best K**
Cocomo81	0.03	0.04	0.04	0.04	0.03	0.03	0.02
Cocomo81e	0.09	0.09	0.09	0.07	0.07	0.03	0.07
Cocomo81o	0.04	0.05	0.05	0.03	0.04	0.04	0.04
Nasa93	0.04	0.05	0.04	0.03	0.03	0.02	0.03
Nasa93c2	0.06	0.05	0.06	0.06	0.05	0.05	0.05
Nasa93c5	0.06	0.07	0.05	0.06	0.06	0.03	0.06
Desharnais	0.05	0.1	0.08	0.05	0.04	0.03	0.04
SDR	0.08	0.07	0.07	0.08	0.05	0.08	0.1
SDR-Banking	0.07	0.14	0.08	0.10	0.05	0.04	0.03
Albrecht	0.07	0.06	0.05	0.06	0.06	0.02	0.07
ISBSG-Banking	0.07	0.1	0.12	0.08	0.08	0.07	0.11

link and single-link clustering may improve the accuracy values. Therefore, in this section we wanted to provide the Pred(25) values for complete and single-link clustering.

The Pred(25) values for all datasets when single-linkage clustering is used are given in Table 6.14. The results in Table 6.14 are in alignment with the results elicited for average-link clustering; that is, TEAK attains the highest Pred(25) values when compared to other k-based methods except two datasets: Albrecht and SDR-Banking. Furthermore, when we compare the Pred(25) values of TEAK for average-link (average-link clustering Pred(25) results are given in parenthesis in Table 6.14) and single-link clustering given in Table 6.14, we see a general trend of improvement in Pred(25) values. The only exception to the general trend of improvement in Pred(25) values is the SDR-Banking dataset. As we remember, SDR-Banking is one of the two most challenging datasets and apart from being challenging we have seen that for that particular dataset average-link clustering works better.

Table 6.14. All Datasets: Mean Pred(25) Values for TEAK and multiple k values with single-link clustering

Dataset	**TEAK**	**k=1**	**k=2**	**k=4**	**k=8**	**k=16**	**k=bestK**
Cocomo81	0.16 (0.13)	0.10	0.10	0.10	0.10	0.13	0.13
Cocomo81e	0.29 (0.24)	0.13	0.23	0.25	0.25	0.29	0.25
Cocomo81o	0.18 (0.14)	0.09	0.14	0.14	0.11	0.11	0.11
Nasa93	0.17 (0.15)	0.12	0.13	0.12	0.13	0.13	0.12
Nasa93c2	0.22 (0.17)	0.08	0.11	0.16	0.14	0.16	0.11
Nasa93c5	0.26 (0.21)	0.18	0.23	0.23	0.23	0.23	0.23
Desharnais	0.36 (0.3)	0.21	0.25	0.30	0.31	0.32	0.32
SDR	0.25 (0.22)	0.13	0.21	0.17	0.17	0.04	0.13
SDR-Banking	0.24 (0.28)	0.20	0.28	0.30	0.34	0.40	0.36
Albrecht	0.21 (0.15)	0.21	0.25	0.25	0.25	0.29	0.23
ISBSG-Banking	0.38 (0.33)	0.25	0.25	0.29	0.32	0.38	0.36

The Pred(25) values for all datasets when complete-linkage clustering is used are given in Table 6.15. As we see from Table 6.15 complete-link clustering improves Pred(25) values when compared to average-link clustering except for two datasets: SDR and SDR-Banking. For SDR and SDR-Banking datasets, the Pred(25) values have dropped from 0.22 to 0.17 and from 0.28 to 0.20 respectively. In all other datasets there is an increase in Pred(25) values. Furthermore, the general trend we observed previously in single and average-link is also valid for complete-link. In other words, TEAK is outperforming all other methods except Albrecht and SDR-Banking dataset.

6.2. Summary of Results for Research Question 1

For ease of use we provide the MRE plots for all the datasets in Figure 6.15. As we can see from the plots in Figure 6.15, TEAK turns out to yield the lowest MRE values in all 11 datasets except SDR-Banking and Albrecht. We also provide the $win - tie - loss$ values for all the datasets in Table 6.16. The values in Table 6.16 also support what we observe in Figure 6.15, that is TEAK outperforms all other methods

Table 6.15. All Datasets: Mean Pred(25) Values for TEAK and multiple k values with complete-link clustering

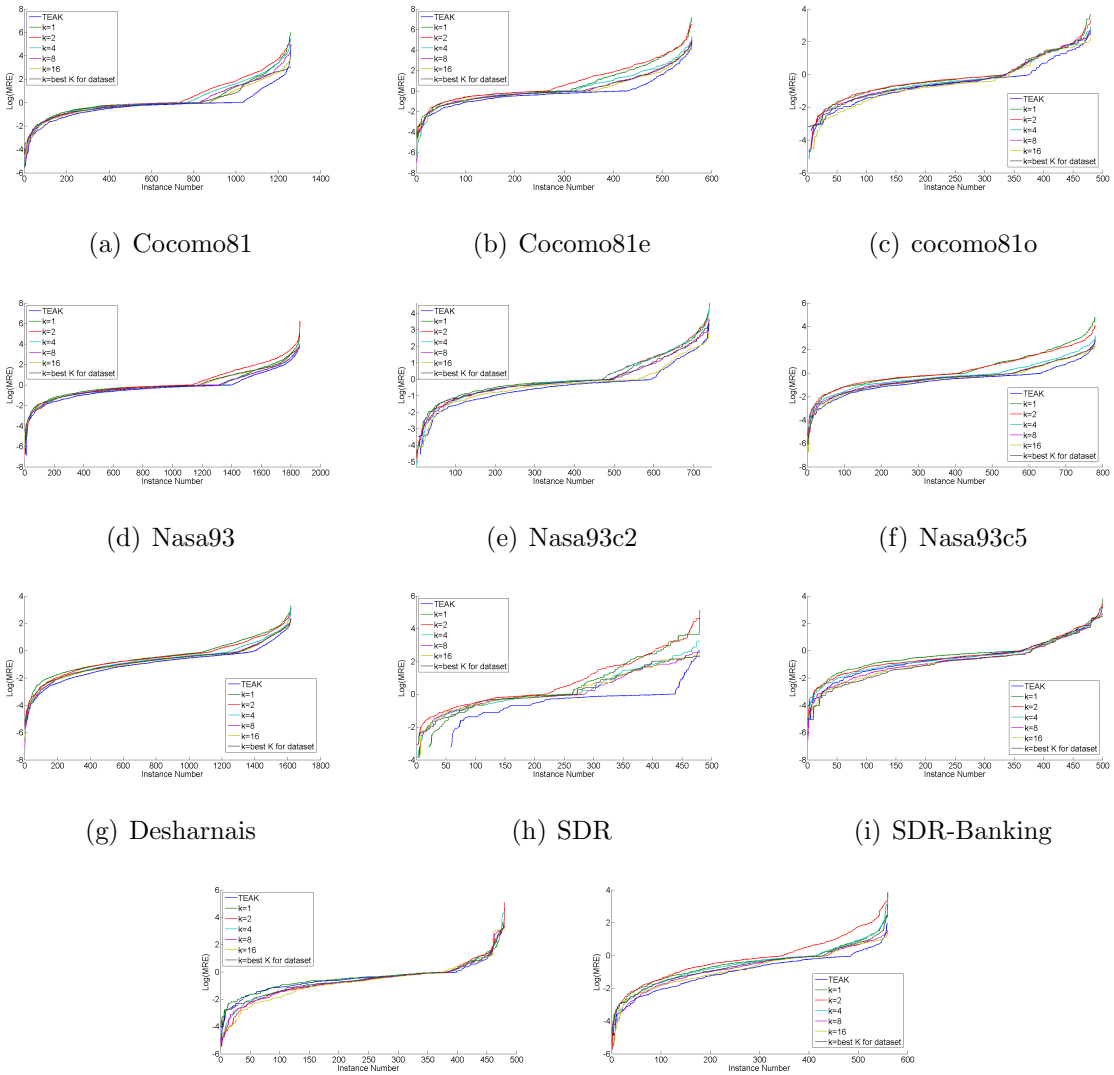
Dataset	**TEAK**	**k=1**	**k=2**	**k=4**	**k=8**	**k=16**	**k=bestK**
Cocomo81	0.16 (0.13)	0.10	0.08	0.12	0.13	0.13	0.11
Cocomo81e	0.23 (0.24)	0.21	0.21	0.25	0.25	0.25	0.25
Cocomo81o	0.18 (0.14)	0.07	0.11	0.11	0.14	0.11	0.11
Nasa93	0.18 (0.15)	0.14	0.14	0.14	0.12	0.15	0.12
Nasa93c2	0.22 (0.17)	0.14	0.14	0.14	0.16	0.15	0.14
Nasa93c5	0.26 (0.21)	0.18	0.15	0.19	0.21	0.23	0.21
Desharnais	0.34 (0.3)	0.27	0.25	0.24	0.28	0.31	0.31
SDR	0.17 (0.22)	0.17	0.13	0.13	0.13	0.06	0.15
SDR-Banking	0.20 (0.28)	0.16	0.24	0.30	0.34	0.40	0.36
Albrecht	0.21 (0.15)	0.17	0.25	0.25	0.27	0.29	0.25
ISBSG-Banking	0.36 (0.33)	0.09	0.09	0.20	0.32	0.36	0.29

in all datasets except SDR-Banking and Albrecht.

As a consequence of all the experimental results provided, we can conclude that with TEAK we have provided a good CBR method for better understanding the characteristics of a dataset and this also provides the answer to our first research question.

6.3. Results for Research Question 2

The second research question in this research was how we could ease the procedure of selecting the best performing number of analogies and whether we could increase the predictive performance while doing that. As a matter of fact, with the utilization of GAC trees in TEAK, we have saved the researcher from the task of choosing the best k value. Since TEAK itself builds up GAC1 and prunes irrelevancies while sending the project instances to GAC2 as well as while building up GAC2, we have left the entire best k selection process to TEAK. Furthermore, apart from being able to choose the number of analogies for each test instance on its own, TEAK outperforms all the other



(a) Cocomo81 (b) Cocomo81e (c) cocomo81o

(d) Nasa93 (e) Nasa93c2 (f) Nasa93c5

(g) Desharnais (h) SDR (i) SDR-Banking

(j) Albrecht (k) ISBSG

Figure 6.15. Log-Mre Values of Datasets for TEAK and multiple k Values

k-based CBR methods.

To illustrate the number of instances that were selected by TEAK while building up GAC2, we provide Figure 6.16, the box plot of those selected instances. As we can see, the fact laying behind the success of TEAK in comparison to all the other static k-based methods is due to the fact that TEAK is able to decide dynamically on the number of analogies to be used. Furthermore, while deciding on the optimum number of analogies to be used in estimation, TEAK is trying to discover the characteristics of the dataset that is best for a particular test instance. From Figure 6.16, we can also see that TEAK can select from a wide range of numbers instead of a fixed k number

and ultimately this results in higher estimation accuracies.

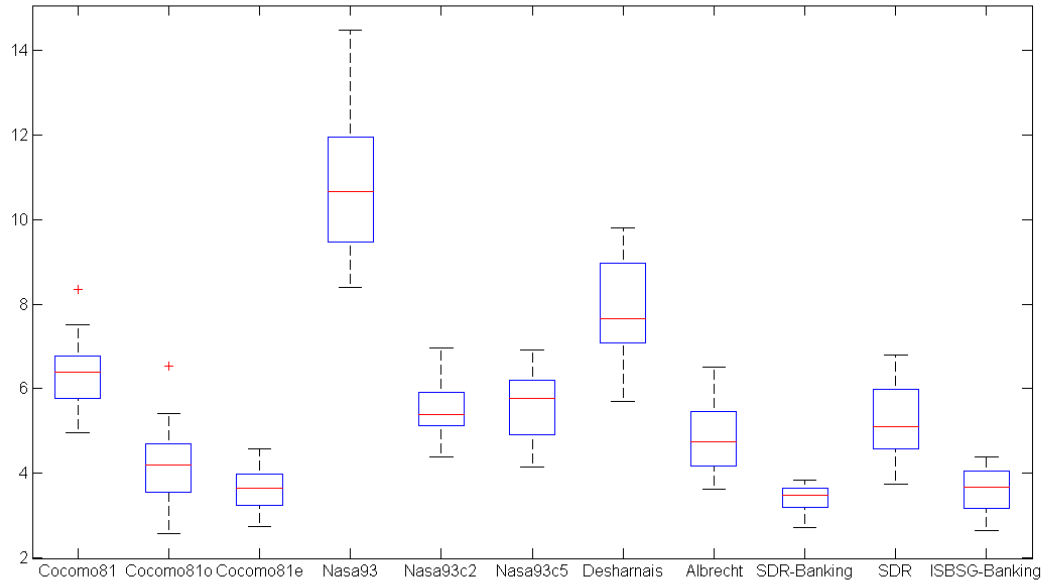


Figure 6.16. Boxplot for the Number of Instances Send to GAC2 Tree

6.4. Threats to Validity

We will address the threats to the validity of our research under four categories:

i. Internal validity
ii. External validity
iii. Construct validity
iv. Statistical validity

Internal validity fundamentally questions to what extent the cause-effect relationship between dependent and independent variables hold. For addressing the threats to internal validity of our results, we used 11 datasets and applied leave-one-out [29] in our experiments so that for each iteration we used a different test instance and a different train set. Furthermore, we have also made 20 runs to eliminate the bias that would otherwise come from the particular test set and train set combination.

External validity, i.e. generalizability of results addresses the extent to which the findings in a particular study are applicable outside the specifications of that study [42]. To ensure the generalizability of our results, we paid extra attention to include as many datasets as possible coming from various resources and used 11 datasets from 3 different sources in our research. Our datasets contain a wide diversity of projects in terms of their sources, their domains and the time period they were developed in. Moreover, the datasets used in this research are composed of software development projects collected from different organizations around the world to be able to generalize our results [6]. We also think that reproducibility of results is an important factor for external validity and we believe that experiment replication is a key factor of experimentation [30]. Therefore, we have purposely selected publicly available datasets and gave as much details as possible regarding the datasets as well as the algorithms used in this research.

Construct validity (i.e. face validity) assures that we are measuring what we actually intended to measure [43]. In our research we are using various criteria for measuring and comparing performance of different models. Those criteria include MRE, Pred(25) and win-tie-loss values, whose detailed definitions as well as calculations are provided in Section 5.3. Comparison of different models in software effort estimation domain by using error based methodologies are criticized for being unreliable [44] [45]. However, a big majority of effort estimation studies use estimation-error based measures for measuring and comparing performances of different methods. The evaluation criteria used in this research also have their roots in estimation error. We have used error based evaluation criteria in our study for three reasons: 1) They are practical options for majority of researchers [46] [47] [3] [48], 2) using error based methods enables our study to be benchmarked with previous effort estimation research and 3) MRE measures the performance of a model on individual instances, which is aligned with our datasets and experimental settings as we use leave-one-out method.

To validate our results statistically we employed statistical significance tests. To mutually compare two classifiers induced on multiple datasets, Wilcoxon signed rank test is suggested [49]. The Wilcoxon signed rank test can be viewed as the non-parametric counterpart of the *t-test* [50]. Therefore, we used Wilcoxon signed rank

test in our study at a confidence level of 95%. Actually the Wilcoxon signed rank test is used as a part of a performance criterion we have used in this research, which is win-tie-loss values. The details regarding how Wilcoxon signed rank test was employed in win-tie-loss calculation is given in Section 5.3.

Table 6.16. Win-Tie-Loss Values for All Datasets

(a) Cocomo81

Algorithm	Win	Tie	Loss	Win-Loss
TEAK	108	132	0	108
k=1	28	148	64	-36
k=2	16	136	88	-72
k=4	20	160	60	-40
k=8	34	170	36	-2
k=16	42	154	44	-2
k=bestK	52	180	8	44

(b) Cocomo81e

Algorithm	Win	Tie	Loss	Win-Loss
TEAK	110	130	0	110
k=1	20	134	86	-66
k=2	6	118	116	-110
k=4	26	166	48	-22
k=8	34	168	38	-4
k=16	44	178	18	26
k=bestK	72	162	6	66

(c) Cocomo81o

Algorithm	Win	Tie	Loss	Win-Loss
TEAK	22	218	0	22
k=1	2	196	42	-40
k=2	2	222	16	-14
k=4	8	228	4	4
k=8	8	232	0	8
k=16	12	228	0	12
k=bestK	8	232	0	8

(d) Nasa93

Algorithm	Win	Tie	Loss	Win-Loss
TEAK	80	160	0	80
k=1	12	122	106	-94
k=2	18	132	90	-72
k=4	28	178	34	-6
k=8	40	182	18	22
k=16	38	194	8	30
k=bestK	44	192	4	40

(e) Nasa93c2

Algorithm	Win	Tie	Loss	Win-Loss
TEAK	42	198	0	42
k=1	16	176	48	-32
k=2	2	166	72	-70
k=4	18	200	22	-4
k=8	28	208	4	24
k=16	26	206	8	18
k=bestK	28	206	6	22

(f) Nasa93c5

Algorithm	Win	Tie	Loss	Win-Loss
TEAK	48	192	0	48
k=1	2	172	66	-64
k=2	6	182	52	-46
k=4	12	200	28	-16
k=8	26	208	6	20
k=16	32	208	0	32
k=bestK	28	210	2	26

(g) Desharnais

Algorithm	Win	Tie	Loss	Win-Loss
TEAK	64	176	0	64
k=1	2	140	98	-96
k=2	14	156	70	-56
k=4	32	192	16	16
k=8	30	202	8	22
k=16	34	200	6	28
k=bestK	30	202	8	22

(h) SDR

Algorithm	Win	Tie	Loss	Win-Loss
TEAK	102	138	0	102
k=1	20	176	44	-24
k=2	2	170	68	-66
k=4	16	202	22	-6
k=8	26	196	18	8
k=16	18	188	34	-16
k=bestK	22	198	20	2

(i) SDR-Banking

Algorithm	Win	Tie	Loss	Win-Loss
TEAK	12	228	0	12
k=1	0	160	80	-80
k=2	12	214	14	-2
k=4	14	212	14	0
k=8	20	220	0	20
k=16	24	216	0	24
k=bestK	26	214	0	26

(j) Albrecht

Algorithm	Win	Tie	Loss	Win-Loss
TEAK	6	234	0	6
k=1	0	230	10	-10
k=2	0	234	6	-6
k=4	0	240	0	0
k=8	4	236	0	4
k=16	6	234	0	6
k=bestK	0	240	0	0

(k) ISBSG-Banking

Algorithm	Win	Tie	Loss	Win-Loss
TEAK	54	186	0	54
k=1	6	154	80	-74
k=2	8	154	78	-70
k=4	18	194	28	-10
k=8	36	196	8	28
k=16	44	196	0	44
k=bestK	34	200	6	28

7. CONCLUSIONS

In this research, to configure CBR systems, we have proposed a novel method called TEAK that makes use of GAC trees. In our research we defined two research questions to address the problems with traditional methods of configuring CBR methods, which can be summarized as follows: 1)Understanding the characteristics of data and 2) determining the number of analogies to be used for better performance. We have shown in Section 6 that previous methods that made use of a fixed number of analogies (fixed-k) on the basis of performance score (MRE or any other error based performance measure) of the overall dataset will eventually sacrifice from the per-instance based optimum k value for a large number projects. Therefore, we proposed utilizing GAC trees that will select small training sets of different sizes, i.e. different number of analogies for each individual test instance. Furthermore, unlike traditional CBR configuration methods, we made use of performance-variance rather than performance score. Therefore, rather than proposing a best-k value a priori as the traditional CBR methods do, what TEAK does is to start with all the training samples in the dataset, learning the dataset to form GAC trees and chopping-off the irrelevant analogies on the basis of variance. As it is reported in Section 6, TEAK has outperformed previously proposed CBR configuration methods.

Apart from addressing the reported issues regarding previous CBR studies in terms of dataset and model, we also meet the methodological problems such as testing only on a limited number of datasets [1] and lacking statistical checks on the results [24]. Therefore, we used various datasets from multiple resources and evaluated our results on the basis of Wilcoxon signed rank test at a 95% confidence level.

7.1. Contributions

We can group the contributions of this research under three headings: 1) Theoretical, 2) practical and 3) methodological. On the following subsections, contributions of this research will be reported under each related subsection.

7.1.1. Theoretical Contributions

The fundamental contribution of this research to academia is a novel CBR calibration method that can actively build itself on the train data and that can choose k neighbours for each test instance dynamically. The main idea behind TEAK was that there is a need for research whose focus is on the data itself rather than the algorithm. In other words we need to *look* at the data rather than looking at the complex algorithms and we need to propose methods that can look at the data, understand it and that can make some sense out of it. In that regard, we believe that our main contribution is a CBR calibration method that was built on the foundations of this idea.

7.1.2. Practical Contributions

Effort estimation in practice is used for decision making and risk evaluation [1] in highly uncertain situations. Therefore, the methods used for effort estimation as well as their results have to be reliable to have a practical value. However, some of the methods proposed previously are criticised for not being reliable due to various reasons such as lacking statistical tests or inaccuracies in how the method can be generalized to different datasets [24]. In this research we proposed a model called TEAK that actually understands data characteristics better than previous fixed-k based CBR methods. The practical contribution of this research is a model called TEAK that provides reliable and generalizable results to practitioners, who would like to use them in real life situations. Furthermore, for this research to have a practical value, we provided all the details regarding how to implement TEAK as well as its results on various datasets subject to statistical significance tests.

7.1.3. Methodological Contributions

Another contribution of this research is the formation of a new dataset in software effort estimation domain. Formation of dataset required 3 months time and included more than 30 meetings with experts from the field. The time spent for those interviews

range from half an hour to 3 hours. Therefore, in this research we do not only contribute with a new dataset but also with hands on experience about measurement as well as data collection.

7.2. Future Work

Going forward, we would like to see how the removal of irrelevant instances in TEAK via utilization of GAC trees relate to the analogies that produce the lowest MRE for each individual instance. That is, we would like to see the relationship between train instances that produced the histograms of Figures 6.1, 6.2, 6.3 and the train instances that are selected by TEAK. We believe that the MRE values achieved by using the optimum number of analogies for each test instance forms a baseline for k-based CBR methods and in our research we have observed that we obtained closer results to this baseline than any other method. However, there is still room to discover between the MRE values of TEAK and the baseline. For instance, the alternative distance measures that we used (single and complete-link clustering) apart from average-link clustering showed us that different distance measures could increase Pred(25) values. As our future work we would like to discover that room between baseline and method results further and lower the error rates to get closer to the baseline.

REFERENCES

1. Kadoda, G., M. Cartwright, and M. Shepperd, "On configuring a case-based reasoning software project prediction system", *UK CBR Workshop, Cambridge, UK*, pp. 1–10, 2000.

2. Heemstra, F., "Software cost estimation", *Information and Software Technology*, pp. 1–14, 1992.

3. Menzies, T., Z. Chen, J. Hihn, and K. Lum, "Selecting Best Practices for Effort Estimation", *IEEE Trans. Softw. Eng.*, Vol. 32, pp. 883–895, 2006.

4. Jø rgensen, M., "A review of studies on expert estimation of software development effort", *Journal of Systems and Software*, Vol. 70, pp. 37–60, February 2004.

5. Kultur, Y., B. Turhan, and A. B. Bener, "ENNA: software effort estimation using ensemble of neural networks with associative memory", *SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pp. 330–338, New York, NY, USA, 2008.

6. Bakir, A., B. Turhan, and A. Bener, "A new perspective on data homogeneity in software cost estimation: a study in the embedded systems domain", 2009.

7. Shepperd, M. and C. Schofield, "Estimating Software Project Effort Using Analogies", *IEEE Trans. Softw. Eng.*, Vol. 23, No. 11, pp. 736–743, 1997.

8. Shepperd, M., C. Schofield, and B. Kitchenham, "Effort estimation using analogy", pp. 170–178, 1996.

9. Boehm, B. W., *Software Engineering Economics*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.

10. Boehm, B. W., Clark, Horowitz, Brown, Reifer, Chulani, R. Madachy, and

B. Steece, *Software Cost Estimation with Cocomo II*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

11. Kruchten, P., "The Rational Unified Process: An introduction", 1999.

12. Menzies, T., O. Elrawas, J. Hihn, M. Feather, R. Madachy, and B. Boehm, "The business case for automated software engineering", pp. 303–312, 2007.

13. Srinivasan, K. and D. Fisher, "Machine Learning Approaches to Estimating Software Development Effort", *IEEE Trans. Softw. Eng.*, Vol. 21, No. 2, pp. 126–137, 1995.

14. Finnie, G. R. and G. E. Wittig, "AI Tools for Software Development Effort Estimation", p. 346, 1996.

15. Jorgensen, M. and M. Shepperd, "A Systematic Review of Software Development Cost Estimation Studies", *IEEE Trans. Softw. Eng.*, Vol. 33, No. 1, pp. 33–53, 2007.

16. Menzies, T. and J. Hihn, "Evidence-Based Cost Estimation for Better-Quality Software", *IEEE Softw.*, Vol. 23, No. 4, pp. 64–66, 2006.

17. Babu, T. and M. Murty, "Comparison of genetic algorithm based prototype selection schemes", *Pattern Recognition*, Vol. 34, p. 523525, 2001.

18. Huang, Y., C. Chiang, J. Shieh, and E. Grimson, "Prototype optimization for nearest-neighbor classification", *Pattern Recognition*, Vol. 35, p. 12371245, 2002.

19. Lipowezky, U., "Selection of the optimal prototype subset for 1-NN classification", *Pattern Recognition Letters*, Vol. 19, p. 907918, 1998.

20. Kirsopp, C. and M. Shepperd, "Making inferences with small numbers of training sets", *IEEE Proc.*, Vol. 149, 2002.

21. Kirsopp, C., M. Shepperd, and R. House, "Case and feature subset selection in case-based software project effort prediction", *Research and development in intelligent systems XIX: proceedings of ES2002, the twenty-second SGAI International Conference on Knowledge Based Systems and Applied Artificial Intelligence*, p. 61, 2003.

22. Li, Y., 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.

23. Fenton, N. E., "Software Metrics: A Rigorous Approach", 1991.

24. Kitchenham, B. and E. Mendes, "Why comparative effort prediction studies may be invalid", pp. 1–5, 2009.

25. Beeferman, D. and A. Berger, "Agglomerative clustering of a search engine query log", *In Knowledge Discovery and Data Mining*, Vol. pages, pp. 407–416, 2000.

26. Guha, S., R. Rastogi, and K. S. Cure, "An efficient clustering algorithm for large databases", *In In Proceedings of ACM SIGMOD International Conference on Management of Data*, Vol. pages, pp. 73–84, 1998.

27. Eisen, M. B., P. T. Spellman, P. O. Brown, and D. Botstein, "Cluster analysis and display of genome-wide expression patterns", *Proceedings of the National Academy of Sciences of the United States of America*, Vol. 95, No. 25, pp. 14863–14868, December 1998.

28. Walter, B., K. Bala, M. Kulkarni, and K. Pingali, "Fast agglomerative clustering for rendering", *IEEE Symposium on Interactive Ray Tracing (RT)*, pp. 2–7, 2008.

29. Alpaydin, E., *Introduction to Machine Learning*, MIT Press, 2004.

30. Juristo, N. and S. Vegas, "Using Differences among Replications of Software Engineering Experiments to Gain Knowledge", *International Symposium on Empirical*

Software Engineering and Measurement, pp. 356–366, 2009.

31. Boetticher, G., T. Menzies, and T. Ostrand, "PROMISE Repository of empirical software engineering data", 2007, `http://promisedata.org/repository`.

32. Menzies, T., D. Port, Z. Chen, J. Hihn, and S. Stukes, "Validation methods for calibrating software effort models", pp. 587–595, 2005.

33. Desharnais, J., *Analyse Statistique de la Productivitie des Projets Informatique a Partie de la Technique des Point des Fonction*, Master's thesis, Univ. of Montreal, 1989.

34. Walkerden, F. and R. Jeffery, "An Empirical Study of Analogy-based Software Effort Estimation", *Empirical Softw. Eng.*, Vol. 4, No. 2, pp. 135–158, 1999.

35. Albrecht, A. and J. Gaffney, "Software function, source lines of code and development effort prediction: A software science validation.", *IEEE Trans. Softw. Eng.*, Vol. 9, pp. 639–648, 1983.

36. Lokan, C., T. Wright, P. R. Hill, and M. Stringer, "Organizational Benchmarking Using the ISBSG Data Repository", *IEEE Softw.*, Vol. 18, No. 5, pp. 26–32, 2001.

37. Bakir, A., *Classification Based Cost Estimation Model for Embedded Software*, Master's thesis, Bogazici University, 2008.

38. Briand, L. C., "A Pattern Recognition Approach for Software Engineering Data Analysis", *IEEE Trans. Softw. Eng.*, Vol. 18, pp. 931–942, 1992.

39. Kitchenham, B. A., L. M. Pickard, S. G. Macdonell, and M. J. Shepperd, "What accuracy statistics really measure", *IEEE Proceedings-Software*, Vol. 148, 2001.

40. Atkinson, K. and M. J. S. ', "The use of function points to find cost analogies"', *in Proc. European Software Cost Modelling Meeting . Ivrea, Italy:*, 1994.

41. Gower, J. C. and G. J. S. Ross, "Minimum Spanning Trees and Single Linkage Cluster Analysis", *Journal of the Royal Statistical Society*, Vol. 18, pp. 54–64, 1969.

42. Milic, D. and C. Wohlin, "Distribution Patterns of Effort Estimations", *Euromicro*, 2004.

43. Robson, C., "Real world research: a resource for social scientists and practitioner-researchers", *Blackwell Publisher Ltd*, 2002.

44. Foss, T., E. Stensrud, B. Kitchenham, and I. Myrtveit, "A Simulation Study of the Model Evaluation Criterion MMRE", *IEEE Trans. Softw. Eng.*, Vol. 29, No. 11, pp. 985–995, 2003.

45. Myrtveit, I., E. Stensrud, and M. Shepperd, "Reliability and Validity in Comparative Studies of Software Prediction Models", *IEEE Transactions on Software Engineering*, Vol. 31, pp. 380–391, 2005.

46. Premraj, R. and T. Zimmermann, "Building Software Cost Estimation Models using Homogenous Data", *ESEM*, 2007.

47. Lum, K., T. Menzies, and D. Baker, "2CEE, A Twenty First Century Effort Estimation Methodology", *ISPA / SCEA*, pp. 12 – 14, 2008.

48. Li, J. and G. Ruhe, "A comparative study of attribute weighting heuristics for effort estimation by analogy", *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, p. 74, 2006.

49. Demsar, J., "Statistical comparisons of classi ers over multiple data sets", *Journal of Machine Learning Research*, Vol. 7, 2006.

50. Jiang, Y., B. Cukic, T. Menzies, and N. Bartlow, "Comparing Design and Code Metrics for Software Quality Prediction", pp. 11–18, 2008.