Noname manuscript No.
(will be inserted by the editor)

Transfer Learning in Effort Estimation

Ekrem Kocaguneli · Tim Menzies · Emilia Mendes

Received: date / Accepted: date

Abstract *Background:* When projects lack sufficient local data to make predictions, they try to transfer information from other projects. How can we best support this process?

Aim: In the field of software engineering, transfer learning has been shown to be effective for defect prediction. This paper checks whether it is possible to build transfer learners for software effort estimation.

Method: We use data on 154 projects from 2 sources to investigate transfer learning between different time intervals and 195 projects from 51 sources to provide evidence on the value of transfer learning for traditional cross-company learning problems.

Results: We find that the same transfer learning method can be useful for transfer effort estimation results for the cross-company learning problem and the cross-time learning problem.

Conclusion: It is misguided to think that: (1) Old data of an organization is irrelevant to current context or (2) data of another organization cannot be used for local solutions. Transfer learning is a promising research direction that transfers relevant cross data between time intervals and domains.

Keywords Transfer Learning, effort estimation, data mining, k-NN

The work was partially funded by NSF grant CCF:1017330 and the Qatar/West Virginia University research grant NPRP 09-12-5-2-470.

E. Kocaguneli and T. Menzies
Lane Department of Computer Science and Electrical Engineering
West Virginia University
Morgantown, WV 26505, USA
E-mail: ekocagun@mix.wvu.edu, tim@menzies.us

Emilia Mendes
Department of Computer Science
University of Auckland
Auckland, New Zealand
E-mail: emilia@cs.auckland.ac.nz

1 Introduction

There is a solid body of evidence that data miners can find interesting and useful patterns from *within-project* data; i.e. the data found within one software organization. This data can take many forms including information from mobile phone app store [Harman et al., 2012]; natural languages requirements documents [Hayes et al., 2006]; logs of software inspections [Menzies et al., 2007, Turhan et al., 2009]; or even the power consumption records associated with software [Hindle, 2012].

The next challenge after *within-project* learning is how to transfer results across *different projects*. *Transfer learning* divides data into *domains* where each domain is a set of examples plus a probability distribution over some class variable (and this distribution must be learned from the examples) [Pan and Yang, 2010]. When we jump from domain1 to domain2, the examples and/or the distribution may change and it is the task of the transfer learner to discover how much of domain1 can be suitably applied to domain2.

Transfer learning can be potentially very useful for software engineering. For example, if a project is using some technology for the first time, it can use transfer learning to acquire valuable insights from another project where developers had more experience in that technology. To facilitate this process, many organizations expend much effort to create repositories of software project data. For example, Rodriguez et al. [D. Rodriguez and Harrison, 2012] report 13 active SE repositories that researchers can access; e.g. PROMISE, BUGZILLA, ISBSG, etc.

These repositories are valuable resources for researchers, since they offer benchmark problems for testing techniques. Better still, if other researchers can access the same repositories, then those benchmark problems encourage replication studies where one researcher tries to repeat, improve, or perhaps even refute the results from another.

While these repositories are useful to researchers, it is an open issue if ***repositories of software project data are valuable to industrial software companies***. This issue must be asked for two reasons: Such repositories *may not predict properties for future projects* and they may be *very expensive to build*:

- Zimmermann et al. report over 600 experiments where data from Project1 was used to build a defect predictor for Project2 [Zimmermann et al., 2009]. In only 3% of those experiments did models learned from Project1 adequately predicted defects for Project2. Other researchers offer similar pessimistic results [Menzies et al., 2012, Menzies et al., 2011, Posnett et al., 2011, Bettenburg et al., 2012]. For example, we have one study where data from six projects is used to predict for defects in a seventh project. The false alarm rate in those predictions was unacceptably high (median values above 65%) [Turhan et al., 2009][1].
- These empirical results are troubling since it can be quite costly to create these repositories. For example, Jairus Hihn from the Jet Propulsion Laboratory [Hihn and Habib-agahi, 1991] reports that NASA spent two million dollars in the early 1990s to build their own repository of project data from 121 NASA software projects developed in the 1970s and 1980s. If we cannot use such expensive repositories to predict properties in future projects, then all that expense is wasted.

It might be argued that, for software engineering, this goal of transfer learning is wrong-headed. After all, software is built by specific people using specific tools for

[1] In the literature, this is known as a *negative transfer effect* [Pan and Yang, 2010] where transfer learning actually makes matters worse.

specific tasks on specific platforms. Given that, perhaps all we can ever hope to do is to monitor some current project until some stable pattern emerges for that specific team, tool, task, and platform. But if that was the case, if all conclusions are local and cannot be transferred to other projects, then:

- Managers would have no basis for planning for future software projects.
- We must conclude that these repositories are ***not*** valuable, at least for the purposes of predicting the properties of new projects.

Recent results in one specific area of software engineering suggest that these concerns may be unfounded. In the specific sub-domain of defect prediction, *relevancy filter* has been shown to allow effective transfer of data across projects. A relevancy filter carefully selects some subset of the training data. For example, weights can be applied to control how we prune away irrelevant training data. Such weights can be learned at the level of instances [Turhan et al., 2009,Ma et al., 2012] or data sets [He et al., 2012]. Alternatively, partitioning methods can be used to find clusters within the training data that are most relevant to the test data [Posnett et al., 2011,Menzies et al., 2011,Menzies et al., 2012,Bettenburg et al., 2012].

These promising results from the area of defect prediction motivate our exploration of other areas of software engineering. This paper explores transfer learning for software effort estimation. Previous results show transferring effort estimation results is a challenging task:

- Kitchenham et al. reviewed 7 published transfer studies in effort estimation[2]. They found that in most (10) cases, transferred data generated worse predictors than using within-project information. This is a worrying finding for, e.g., start-up companies since it means they will not be able to make adequate effort estimates until after completing numerous local projects (and, in the meanwhile, they run the risk of projects failing due to inadequate effort allocations).
- In a similar result, Ye et al. report that the tunings to Boehm's COCOMO model have changed radically for new data collected in the period 2000 to 2009 [Yang et al., 2011].

Note that effort estimation is a different problem to defect prediction:

- While defect data sets store information on hundreds to millions of methods,
- Effort data sets are smaller, often they are just a few dozen records.

Since the data sets are so different, we must use different methods to transfer effort estimation data. In prior work [Kocaguneli et al., 2012], we have found that a self-tuning analogy-based effort estimation method called TEAK out-performs other approaches such as linear regression, neural networks, and traditional analogy-based reasoners. Like the above work on defect prediction, TEAK uses relevancy filtering. TEAK explores outwards from each test instance, taking care not to enter "suspicious" regions where consensus breaks down and the training data offers wildly contradictory conclusions. To optimize that process, TEAK first builds a dendogram (a tree of clusters of project data), then computes the variance of the conclusions of the examples in each sub-tree.

This paper uses TEAK as laboratory for studying transfer learning in effort estimation. Using TEAK, we will explore the following research questions.

[2] Terminology note: Kitchenham et al. called such transfers "cross-company" learning.

RQ1: Is transfer learning possible outside the area of defect prediction?

Our studies will divide effort data into numerous stratifications (e.g. different companies, different development sites, different languages, etc). All this data will then be loaded into TEAK. Our software will then group together similar data seen in different stratifications.

RQ2: Is transfer learning effective for effort estimation?

We will compare estimates learned from within a single stratification to those transferred across different stratifications.

RQ3: How useful are manual divisions of the data?

TEAK has a choice about where it finds its training data: Either in one stratification or in many. Hence it can assess the relative merits of automated vs manual groups of data. Automated groups are generated by methods like TEAK. The alternative to automatic grouping is manual grouping via *delphi localization*; i.e. the practice where (a) some human expert offers a division of the data then (b) effort models are built from just the data within each such localization[3].

RQ4: Does transfer learning for effort estimation work across time as well as space?

We find that transfer learning is a unifying framework for two, previously distinct, research themes in software engineering. Turhan discusses the problem of *data set shift* [Turhan, 2012] where, within one project, something has changed and the old data for that project no longer applies. This issue of changes within one project is not discussed in the *cross-company learning* literature in SE [Kitchenham et al., 2007, Turhan et al., 2009] since cross-company learning usually focuses on, say, transferring data across two divisions of one company found in two different physical locations. Note that, from the perspective of transfer learning, both problems are the same; i.e. the transfer data from domain1 to domain2 regardless of whether the domains are:

- from the same project at different times (i.e. the data set shift problem);
- or from different projects at similar times in different places (i.e. the cross company learning problem).

This leads to the following conjecture: A transfer learner that works for cross-company learning can also be applied to transferring past data to current projects. To test that conjecture, this paper applies the same transfer learner *without modification* to traditional cross-company learning problems *and* data set shift problems.

RQ5: Are repositories of software project data valuable to industrial companies?

Based on the above, we will make an assessment of the question that originally motivated this paper.

RQ6: Does the history of software engineering have relevant lessons for today?

[3] Examples of delphi localizations come from Boehm [Boehm, 1981] and Petersen & Wohlin [Petersen and Wohlin, 2009]. Boehm divided software projects into one of the "embedded", "semi-detached" or "organic" projects and offered different COCOMO-I effort models for each. Petersen & Wohlin offer a rich set of dimensions for contextualizing projects (processes, product, organization, etc).

We know of many researchers and managers that are so enamoured with the fast pace of change in SE that they are willing to throw out conventional wisdom in favor of new ideas, even when those ideas are still immature and not fully investigated. This "everything must be new" mentality often effects our own work. As data mining researchers, we often struggle to find appropriate data for checking our proposed prediction methods. Previously, we have encountered some resistance to the use of seemingly "out-dated" data sets from last century such as COC-81 and NASA-93 for a 21^{st} century paper on effort estimation. Therefore, in this paper, we check if any parts of older data sets have any relevancy on later projects.

Note that, from an economic perspective, this last point is quite important. Collecting data can be very expensive (we saw above that the NASA-93 was the result of a two million dollar project review exercise conducted by NASA). If any organization goes to all the trouble of building such repositories, it seems important and appropriate to understand the life expectancy of that data.

1.1 Contributions and Structure

In summary, the contributions of this study are:

- Proposing transfer learning as a solution to the important problem of local data issues for effort estimation.
- Empirically showing that effort estimation data can be successfully transfered without performance loss across time and space. To the best of our knowledge, this is the first report in the effort estimation literature of a successful transfer learning application that can effectively transfer data across time and space.
- Evaluating the success of transfer learning on recent proprietary as well as public data sets; hence, providing evidence for practitioners as well as benchmark availability for further research.

This paper extends a prior publication as follows:

- In 2011, Kocguenli et al. offered a limited study on just cross-company learning with 8 data sets [Kocaguneli and Menzies, 2011a]. This paper doubles the number of case studies *as well as* exploring issues of data set shift. That is, that prior publication explored learning across space while here we explore transfer across time *and* space.
- Furthermore, the data used in that prior study is somewhat dated while this paper uses more recent data from organizes building Web-based applications.
- This research uses more evaluation criteria including a recently proposed error measure [Shepperd and MacDonell, 2012].
- Lastly, that prior aforementioned work did not recognize the connection of its research to transfer learning. This paper makes that connection via a more extensive literature review.

The rest of this paper is structured as follows. After some background notes, TEAK is evaluated on the proprietary data sets of 8 Web companies from the Tukutuku [Mendes and Mosley, 2008] data base (for transfer between domains) as well as publicly available NASA data sets called Cocomo81[4] and Nasa93[5] (for transfer between time intervals). In the experimentation, each test instance is evaluated in two different scenarios:

[4] http://goo.gl/WxGXv
[5] http://goo.gl/ioXDy

– In the first scenario, the test instance is allowed to use only within training data (i.e. restricted to its own domain or time interval).
– In the second scenario, the test instance is allowed to transfer data or use within-project data (i.e. it is allowed to transfer instances from other domains or time intervals). This second scenario study shows that there is a considerable amount of cross data that can be successfully transferred across time and space.

The performance of the test instances in both scenarios are compared according to 8 different error measures (details in §3.2) subject to Wilcoxon test (at 95% confidence). In 6 out of 8 companies, transfer learning resulted in performance that is statistically identical to estimates generated from within-data experiments. In all cases of the transfer learning between time intervals, the within and transferred data performance were statistically the same.

2 Related Work

In this paper, we will refer to transfer learning and software effort estimation as TL and SEE (respectively).

2.1 Transfer Learning

A learning problem can be defined by:

– a specific domain D, which consists of a feature space and a marginal distribution defining this space;
– and a task T, which is the combination of a label space and an objective estimation function.

TL allows for the training and test data to have different domains and tasks [Ma et al., 2012]. According to Jialin et al. TL can be formally defined as follows [Pan and Yang, 2010]: Assuming we have a source domain D_S, a source task T_S, a target domain D_T and a target task T_T; TL tries to improve an estimation method in D_T using the knowledge of D_S and T_S. Note that the assumption in the above definition is that $D_S \neq D_T$ and $T_S \neq T_T$. There are various subgroups of TL, which define the relationship between traditional machine learning methods and various TL settings, e.g. see Table 1 of [Pan and Yang, 2010]. SEE transfer learning experiments have the same task but different domains, which places them under the category of transductive TL [Arnold et al., 2007].

There are 4 different approaches to TL [Pan and Yang, 2010]: instance-transfer (or instance-based transfer) [Foster et al., 2010], feature representation transfer [Lee et al., 2007], parameter-transfer [Gao et al., 2008] and relational-knowledge transfer [Mihalkova et al., 2007]. The TL approach of the estimation method used in this research corresponds to instance-transfer. The benefits of instance-transfer learning are used in various research areas, e.g. Ma et al. use TL for cross-company defect prediction, where they use a weighted Naive Bayes classifier [Ma et al., 2012]. Other research areas that benefit from instance-transfer are text classification [Dai et al., 2007], e-mail filtering [Zhang et al., 2007] and image classification [Wu and Dietterich, 2004].

TL between different time intervals has been paid very little attention in SEE. To the best of our knowledge, the only work that has previously questioned TL between different time frames is that of Lokan et al. [Lokan and Mendes, 2009a, Lokan and Mendes, 2009b]. In [Lokan and Mendes, 2009b] Lokan and Mendes found out by using chronological sets of instances that time frame divisions of instances did not affect prediction accuracy. In [Lokan and Mendes, 2009a], they found out that it is possible to suggest a window size of a time frame of past instances, which can yield performance increase in estimation. They also note that the size of the window frame is data set dependent. Our research builds on the prior findings to provide evidence of knowledge transfer through both space and time.

The prior results on the performance of TL (a.k.a. cross-company learning in SEE) are unstable. In their review, Kitchenham et al. [Kitchenham et al., 2007] found equal evidence for and against the value of TL in SEE. Out of the 10 studies reviewed by Kitchenham et al., 4 studies favored within data, another 4 studies found that transferring data is not statistically significantly worse than within data, and 2 studies had inconclusive results. In the field of defect prediction, Zimmermann et al. studied the use of transferring data [Zimmermann et al., 2009]. Zimmermann et al. found that predictors performed worse when trained on cross-application data than from within-application data. From a total of 622 transfer and within data comparisons, they report that within performed better in 618 cases. Recently Ma et al. defined using data across other domains in the research field of defect prediction as a TL problem [Ma et al., 2012]. Ma et al. propose a Naive Bayes variant, so called Transfer Naive Bayes (TNB), so as to use all the appropriate features from the training data. TNB is proposed as an alternative TL method for defect prediction when there are too few training data. According to Ma et al. data transferring problem of defect prediction corresponds to an inductive TL setting; where source and target tasks are the same, yet source and target domains are different. The inductive TL methods are summarized as either instance transfer or feature transfer [Huang et al., 2007]. The current literature of TL in defect prediction as well as SEE focuses on instance transfer.

Turhan et al. compared defect predictors learned from transferred or within data. Like Zimmermann et al., they found that transferring *all* data leads to poor estimation method performance (very large false alarm rates). However, after *instance selection* pruned away irrelevant data during TL, they found that the estimators built on transferred data were equivalent to the estimators learned from within data [Turhan et al., 2009]. Motivated by Turhan et al. [Turhan et al., 2009], Kocaguneli et al. [Kocaguneli et al., 2010] used *instance selection* as a pre-processor for a study of TL in SEE, where test instances are allowed to use only transferred or only within data. In a limited study with three data sets, they found that through instance selection, the performance differences in the predictors trained on transferred or within data were not statistically significant. This limited study was challenged by Kocaguneli et al. in another study that uses 8 different data sets [Kocaguneli and Menzies, 2011b]. The results were identical: performance differences of within and transferred data are not significant.

2.3 Data Set Shift

A repeated result in software engineering is that the conclusions that hold in one context do not hold in another. For example, Keung et al. investigated 90 SEE methods induced on 20 data sets [Keung et al., 2012]. Their results behaved in accordance with the prediction of Shepperd et al. [Shepperd and Schofield, 1997], i.e. changing conditions (different data sets and/or error measures) change the ranking of methods.

Turhan offers a formal treatment for this problem of "context change" in software engineering [Turhan, 2012]. He discusses *data set shift* which appears when training and test joint distributions are different[6]. Turhan investigates different types the data set shift issues as proposed by Storkey [Storkey, 2009]:

- Covariate shift: The covariates in test and train sets differ;
- Prior probability shift: Prediction model is found via Bayes Rule, yet the distribution of the dependent variable differ for training and test sets;
- Sample selection bias: The training and test sets are selected from populations with different characteristics, e.g. training set coming from a higher maturity level company being used on a test set of a lower maturity level company;
- Imbalanced data: Certain event (or class) types of interest are rarer compared to other events (or classes);
- Domain shift: The cases where the method of measurement (e.g. performance measures, size measures) changes between training and test conditions.
- Source component shift: Parts of data come from different sources with particular characteristics in varying sizes, e.g. data sets that are collection of instances collected in different companies or time frames.

Turhan continues his discussion by introducing techniques to handle data set shift problems in software engineering, which are grouped under two main groups: Instance-based techniques and distribution-based techniques. The former group of techniques aim at handling instances (through outlier removal [Turhan et al., 2009], relevancy filtering [Kocaguneli and Menzies, 2011b] or instance weighting [Zhang and Sheng, 2004]), whereas the latter group of techniques aim at regulating the distribution of instances to train and test sets (through stratification [Turhan, 2012], cost curves [Jiang et al., 2008] and mixture models [Alpaydin, 2010]).

Quoting from Storkey: "Dataset shift and transfer learning are very related." Data set shift is a specific case of transfer learning. Transfer learning deals with the cases, where there are multiple training scenarios that are partially related and are used to predict in one specific scenario. For example, in the case of our research multiple training scenarios are either data from different companies (transfer of data in space) or data from different time frames (transfer of data in time). Dataset shift is the specific case, where there are only two scenarios (training and test sets) and one of them has no training targets [Storkey, 2009]. Turhan's mapping of the data set shift types to software engineering is an excellent example of specific transfer learning issues that are becoming imminent.

In this research, we provide a more general treatment of the transfer learning and propose that our earlier work that we called "cross company learning" [Kocaguneli

[6] Note that the literature contains numerous synnonyms for data set shift including "concept shift" or "concept drift", "changes of classification", "changing environments", "contrast mining in classification learning","fracture points" and "fractures between data". We will use the term as defined in the above text.

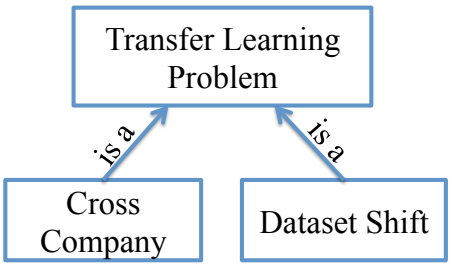


Fig. 1: Prior studies of cross company and dataset shift studies are specific cases of transfer learning.

and Menzies, 2011a] as well as specific data set issues are in fact particular cases of transfer learning (see Figure 1). One of the main ideas of this research is that separate problems of cross company and time frame estimation as well as data set shift problems are the same. Hence, instead of proposing different solutions for each specific case of each specific problem, we should be looking for general transfer learning solutions that work for multiple cases in both problem types.

3 Methodology

3.1 Dataset

This study uses the Tukutuku data base for TL between domains [Mendes and Mosley, 2008]. Tukutuku brings together data sets coming from a high number of companies. The version used in this research is composed of a total of 195 projects developed by a total of 51 companies. However, not all the companies in the data set are useful for TL analysis. We eliminated all the companies with less than 5 projects, which yielded 125 projects from 8 companies. The abbreviations used for the 8 companies that were selected and their corresponding number of projects are given in Figure 2.

The Tukutuku data base is an active project, which is maintained by Emilia Mendes. The data set includes information collected from completed Web projects [Mendes et al., 2005]. These projects come from a total of 10 different countries around the world [Corazza et al., 2010]. The majority of the projects in Tukutuku data base are new development (65%), whereas the remaining ones are enhancement projects. Tuku-

Company	# of Projects
tuku1	14
tuku2	20
tuku3	15
tuku4	6
tuku5	13
tuku6	8
tuku7	31
tuku8	18

Fig. 2: The abbreviations that will be used for selected companies and the corresponding number of projects.

tuku data base is characterized by a total of 19 independent variables and a dependent variable. The dependent variable is the total effort in person hours used to develop an application. The abbreviations as well as the meanings of the independent variables are as follows:

- Typeproj: Project type, enhancement or new
- nlang: Number of programming languages used in the project.
- DocProc: Whether or not projects followed defined and documented processes
- ProImpr: Whether or not project team involved in a process improvement program
- Metrics: Whether or not project team is part of a software metrics program
- DevTeam: Size of the development team
- TeamExp: Average team experience with the employed development language(s)
- Accuracy: The procedure used to record effort data
- TotWP: Total number of Web pages (new and reused)
- NewWP: Total number of new Web pages
- TotImg: Total number of images (new and reused)
- NewImg: Total number of new images created
- Fots: Number of features reused without any adaptation
- HFotsA: Number of reused high-effort features/ functions adapted
- Hnew: Number of new high-effort features/functions
- totHigh: Total number of high-effort features/ functions
- FotsA: Number of reused low-effort features adapted
- New: Number of new low-effort features/functions
- totNHigh: Total number of low-effort features/functions

So as to see the TL performance and selection tendency between time intervals, we used 2 data sets: Cocomo81 and Nasa93. Each of these data sets are divided into 2 subsets of different time periods. The subsets of Cocomo81 are: coc-60-75 and coc-76-rest, where *coc* stands for Cocomo81, *60-75* stands for projects developed from 1960 to 1975 and *76-rest* stands for projects developed from 1976 onwards. It is possible to have different divisions of the data depending on different time frames. Our selection selects these particular divisions of time periods so that both subsets span a certain amount of time (e.g. more than a decade) and yet have at least 20 instances. The subsets of nasa93 are: nasa-70-79 and nasa-80-rest. The naming convention is similar to that of Cocomo81 subsets: *nasa* stands for Nasa93 dataset, *70-79* stands for projects developed in the time period of 1970 to 1979 and *80-rest* stands for projects developed from 1980 onwards. The details of these projects are provided in Figure 3.

Dataset	**Features**	**Size**	**Description**
coc-60-75	17	20	Nasa projects between 1960 and 1975
coc-76-rest	17	43	Nasa projects from 1976 onwards
nasa-70-79	17	39	Nasa projects between1970 and 1979
nasa-80-rest	17	54	Nasa projects from 1976 onwards
		Total: 156	

Fig. 3: Data sets for transfer learning over time period.

3.2 Performance Measures

There are multiple performance measures (a.k.a. error measures) used in SEE. In this research, we use a total of 8 error measures. Performance measures aim to measure the success of a prediction. For example, the absolute residual (AR) is the absolute difference between the predicted and the actual:

$$AR_i = |x_i - \hat{x_i}| \tag{1}$$

(where $x_i, \hat{x_i}$ are the actual and predicted value for test instance i). We use a summary of AR through taking the mean of AR, which is known as Mean AR (MAR).

The Magnitude of Relative Error measure a.k.a. MRE is a very widely used performance measure for selecting the best effort predictor from a number of competing software prediction models [Shepperd and Schofield, 1997,Foss et al., 2003]. MRE measures the error ratio between the actual effort and the predicted effort and is expressed by the following equation:

$$MRE_i = \frac{\mid x_i - \hat{x_i} \mid}{x_i} = \frac{AR_i}{x_i} \tag{2}$$

A related measure is MER (Magnitude of Error Relative to the estimate [Foss et al., 2003]):

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

The overall average error of MRE can be derived as the Mean or Median Magnitude of Relative Error measure (MMRE and MdMRE, respectively):

$$MMRE = mean(allMRE_i) \tag{4}$$

$$MdMRE = median(allMRE_i) \tag{5}$$

A common alternative to MMRE is PRED(25), which is defined as the percentage of successful predictions falling within 25% of the actual values, and can be expressed as follows, where N is the dataset size:

$$PRED(25) = \frac{100}{N} \sum_{i=1}^{N} \begin{cases} 1 \text{ if } MRE_i \leq \frac{25}{100} \\ 0 \text{ otherwise} \end{cases} \tag{6}$$

For example, PRED(25)=50% implies that half of the estimates fall within 25% of the actual values [Shepperd and Schofield, 1997].

Other performance measures used here are Mean Balanced Relative Error (MBRE) and the Mean Inverted Balanced Relative Error (MIBRE), both suggested by Foss et al. [Foss et al., 2003]:

$$MBRE_i = \frac{|\hat{x_i} - x_i|}{min(\hat{x_i}, x_i)} \tag{7}$$

$$MIBRE_i = \frac{|\hat{x_i} - x_i|}{max(\hat{x_i}, x_i)} \tag{8}$$

The above mentioned performance measures are selected due to their wide use in the SEE research. However, none of these error measures are devoid of problems. For

instance, MRE-based error measures have been criticized due to their asymmetry [Foss et al., 2003]. This criticism applies to MMRE, MdMRE and Pred(25). The matter of providing an in depth discussion about error measures is out of our scope in this paper. Rather than going into the debate of which error measure is better than the others, we evaluated our results subject to a total of 8 error measures. A recent study by Shepperd et al. provides an excellent discussion of the error measures [Shepperd and Macdonell, 2011]. In this study Shepperd et al. propose a new unbiased error measure called Standardized Accuracy (SA), which is based on the mean absolute error (MAE). SA's equation is as follows:

$$SA = 1 - \frac{MAE_{P_i}}{\overline{MAE_{P_0}}} \tag{9}$$

MAE_{P_i} is defined to be the MAE of the estimation method P_i. $\overline{MAE_{P_0}}$ is the mean of a large number of (in our case 1000) random guessing. In the random guessing procedure a training instance is randomly chosen with equal probability from the training set (with replacement) and its effort value is used as the estimate of the test instance. SA gives us an idea of how good an estimation method is in comparison to random guessing. Since the term MAE_{P_i} is in the nominator, the higher the SA values, the better an estimation method.

3.3 Instance Selection and Retrieval

One of our goals in this research is to find the selection tendency of test instances, i.e. the percentages of training instances (with respect to the training set size) selected from within or transferred data. This investigation will help us identify what percent of the data across domain or time frame are transferred for a test instance, given the chance that it can select either within or transferred instances. If test instances were to select mostly the within domain or within time interval instances (i.e. transferred instances are not much used), then there would not be much need for a study like this; because, within sources would turn out to be the most relevant instances to test instances. However, as we will see in our results section, that is not the case.

Inspecting the selection tendency of an estimation method for test instances makes sense only if the estimation method is a state-of-the-art learner. Because, inspection of the selected instances has its merits, provided that it leads to estimates that are as good as that of the best-of-breed estimation methods. Therefore, we selected to use a variance-based instance selection method that was previously shown to successfully work on TL experiments (across space) on public data sets. In this section we briefly introduce the method used as a TL solution, called TEAK [Kocaguneli et al., 2012]. Then we quote prior research to show that TEAK's performance is comparable to or better than various other estimation methods.

3.3.1 ABE0

Analogy-based estimation (ABE) methods generate their estimates by using a data base of past projects. For a *test* project, ABE methods retrieve analogies from a database of past projects. Then the effort values of the retrieved analogies are adapted into an estimate. We use ABE methods in this study since 1) they are widely investigated

methods in the literature [Mendes et al., 2003, Chang, 1974, Keung, 2008, Li et al., 2009, Kadoda et al., 2000, Kocaguneli et al., 2010, Kocaguneli et al., 2012], 2) they are particularly helpful for TL studies as they are based on distances between individual project instances.

There is a high number of design options concerning ABE methods such as the distance measure for nearness [Mendes et al., 2003], adaptation of analogy effort values [Mendes et al., 2003], row processing [Chang, 1974, Keung, 2008], column processing [Keung, 2008, Li et al., 2009] and so on. For example, Keung et al. show that the number of different design options can easily lead to more than 6000 ABE variants [Keung et al., 2012]. Here we define ABE0 that is a *baseline* ABE method that combines the methods used in Kadoda & Shepperd [Kadoda et al., 2000], Mendes et al. [Mendes et al., 2003], and Li et al. [Li et al., 2009]:

- Input a database of past projects
- For each test instance, retrieve k similar projects (analogies).
  - For choosing k analogies use a similarity measure.
  - Before calculating similarity, scale independent features to 0-1 interval so that higher numbers do not dominate the similarity measure.
  - Use a feature weighting scheme to reduce the effect of less informative features.
- Adapt the effort values of the k nearest analogies to come up with the effort estimate.

ABE0 uses the Euclidean distance as a similarity measure, detailed in Equation 10, where w_i corresponds to feature weights applied on independent features. The ABE0 framework does not favor any features over the others, therefore each feature has equal importance in ABE0, i.e. $w_i = 1$. For adaptation, ABE0 takes the median of k projects.

$$Distance = \sqrt{\sum_{i=1}^{n} w_i(x_i - y_i)^2} \tag{10}$$

3.3.2 TEAK

TEAK is a *variance-based* instance selector that discards training data associated with regions of high dependent variable (effort) variance [Kocaguneli et al., 2012]. TEAK is based on the locality principle, which states that instances that are close to one another in space according to a distance measure (e.g. Euclidean distance measure) are similar instances and should have similar dependent variable values. A high variance region, where similar instances have very different effort values (hence the high variance) violates the locality assumption and is pruned away by TEAK [Kocaguneli et al., 2012]. TEAK augments ABE0 with instance selection and an indexing scheme for filtering relevant training examples. In summary, TEAK is a two-pass system:

- Pass 1 prunes training instances implicated in poor decisions (instance selection);
- Pass 2 retrieves closest instances to the test instance (instance retrieval).

In the first pass, training instances are combined using greedy-agglomerative clustering (GAC), to form an initial cluster tree that we call GAC1; e.g. see Figure 4. Level zero of GAC1 is formed by leaves, which are the individual project instances. These instances are greedily combined (combine the two closest instances) into tuples to form the nodes of upper levels. The variance of the effort values associated with

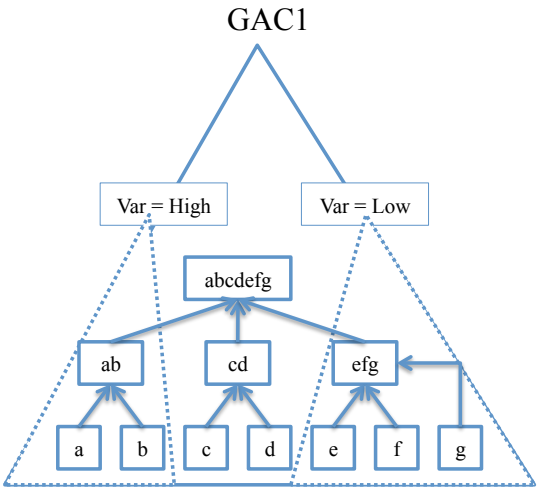


Fig. 4: A sample GAC tree with regions of high and low variance (dashed triangles). GAC trees may not always be binary. For example here, leaves are odd numbered, hence node "g" is left *behind*. Such instances are pushed *forward* into the closest node in the higher level. For example, "g" is pushed forward into the "e+f" node to make "e+f+g" node.

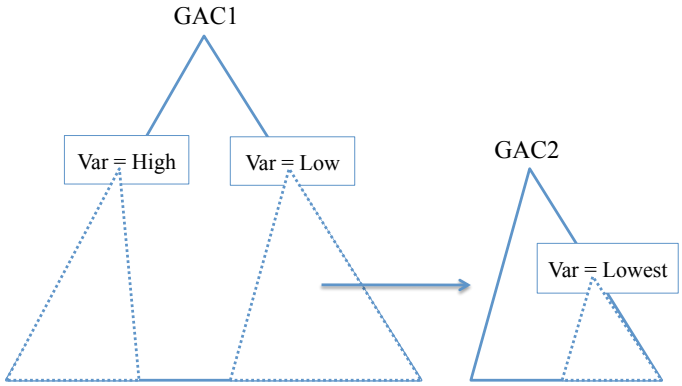


Fig. 5: Execution of TEAK on 2 GAC trees, where tree on the left is GAC1 of Figure 4 and the one on the right is GAC2. The instances in the low variance region of GAC1 are selected to form GAC2. Then test instance traverses GAC2 until no decrease in effort variance is possible. Wherever the test instance stops is retrieved as the subtree to be used for adaptation (var=lowest labeled, dashed triangle of GAC2).

each sub-tree (the performance variance) is then recorded and normalized: $min..max$ to 0..1. The high variance sub-trees are then pruned, as these are the sub-trees that would cause an ABE method to make an estimate from a highly variable instance space. Hence, pass one prunes sub-trees with a variance in the vicinity of $rand() * \alpha\%$ of the maximum variance seen in any tree, where $rand()$ gives a normal random value from 0-1 interval. After some experimentation, we found that $\alpha = 10$ leads to estimates with lowest errors.

The leaves of the remaining sub-trees are the *survivors* of pass one. They are filtered to pass 2 where they are used to build a second GAC tree (GAC2). GAC2 is generated in a similar fashion to GAC1, then it is traversed by test instances that are moved from root to leaves. Unlike GAC1, this time variance is a decision criterion for the movement of test instances: If the variance of the current tree is larger than its sub-trees, then continue to move down; otherwise, stop and retrieve the instances in the current tree as the analogies. TEAK is a form of ABE0, so its adaptation method is the same, i.e. take the median of the analogy effort values. A simple visualization of this approach is given in Figure 5.

We use TEAK in this study since, as shown by the leave-one-out experiments of Kocaguneli et al. [Kocaguneli et al., 2012], its performance is comparable, or even better, than other commonly-used effort estimators including neural networks (NNet) and linear regression (LR). For a complete analysis of TEAK compared to NNet, LR as well as various ABE0 methods please refer to Figure 7 of [Kocaguneli et al., 2012].

That study evaluated different SEE methods using the $win - loss$ calculation of Figure 6. We first check if two distributions i, j are statistically different according to the Wilcoxon test. In our experimental setting, i, j are arrays of performance measure results coming from two different methods. If they are not statistically different, then they are said to *tie* and we increment tie_i and tie_j. On the contrary, if they are different, we updated win_i, win_j and $loss_i, loss_j$ after a numerical comparison of performance measures. The related pseudocode is given in Figure 6. In order to reduce any possible bias due to a particular experimental setting, for every experiment 20 runs are made.

In the study of Kocaguneli et al. [Kocaguneli et al., 2012], TEAK always performed better than the other ABE0 methods, and mostly performed better than neural nets. TEAK's only near-rival was linear regression but (using Figure 6) TEAK was ranked top nearly twice as much as linear regression.

3.4 Experimentation

The goals of the experiments carried out herein can be summarized as follows:

1. To answer whether TL can enable the use of data from other organizations as well as from other time intervals: We employ TEAK as a TL method and compare its performance when trained from just within data versus when trained from a combination of transferred and within data.
2. The *retrieval tendency* questions the tendency of a *within* test instance to retrieve *within* or *transferred* data. In other words, given the chance that a test instance had access to *within* and *transferred* data at the same time, what percentage of every subset would be retrieved into k analogies used for estimation?

The first goal searches for a very important topic in SEE: Is it possible to use TL so that we can transfer training instances between time intervals and different companies? If yes, then how good of a performance would TL yield, i.e. can we still attain performance values as good as using training data from within source (same domain or time interval). The second goals is somewhat complementary to the first one. Second goal questions whether test instances make use of an important amount of transfered data or whether the transfered data from across different domains is so small that it is not really worth investigating.

3.4.1 Performance Comparison

With regard to performance comparison we have two settings: *Within* and *transfer*. In the *within* data setting, only the *within* source is used as the dataset, and a testing strategy of leave-one-out cross-validation (LOOCV) is employed. LOOCV works as follows: Given a *within* dataset of T projects, 1 project at a time is selected as the test and the remaining $T-1$ projects are used for training, so eventually we have T predictions. The resulting T predictions are then used to compute 8 different performance measures defined in §3.2.

Transfer setting uses one instance at a time (since we use LOOCV) from the *within* data as the test set and the *combination of the remaining within instances and all the transferred* data as the training set. In this setting an estimate for each test instance is found via the transfer of the analogies from the training set by TEAK. Ultimately we end up with T predictions adapted from the analogies that are transferred from a training set of transferred and within instances. Finally, the performances under *within* and *transfer* settings are compared. For that purpose, we use both mere performance values as well as win-tie-loss statistics.

3.4.2 Retrieval Tendency

For retrieval tendency experiments we mark every *within* and *transferred* instance in the training set and let the test instance choose analogies from both groups of training

$win_i = 0$, $tie_i = 0$, $loss_i = 0$
$win_j = 0$, $tie_j = 0$, $loss_j = 0$
if Wilcoxon($Perf_i$, $Perf_j$) says they are the same **then**
  $tie_i = tie_i + 1$;
  $tie_j = tie_j + 1$;
else
  if mean or median($Perf_i$) < median($Perf_j$) **then**
    $win_i = win_i + 1$
    $loss_j = loss_j + 1$
  else
    $win_j = win_j + 1$
    $loss_i = loss_i + 1$
  end if
end if

Fig. 6: Pseudocode for win-tie-loss calculation between methods i and j w.r.t. performance measures $Perf_i$ and $Perf_j$. If $Perf_i$ and $Perf_j$ are measures like MMRE, MdMRE or MAR, lower values are better, whereas for performance measures like Pred(25) higher values are better.

instances. Note that retrieved analogies are the unique training instances in the lowest-variance region of GAC2 (see Figure 5). In this setting our aim is to see what percentage of *within* and *transferred* data would appear among retrieved k analogies, i.e. how much of transferred data is relevant? The retrieval percentage is the average ratio of instances retrieved in analogies to the total size of the training set:

$$Percentage = \frac{NumberOfRetrievedAnalogies}{TrainingSetSize} * 100 \quad (11)$$

4 Results

4.1 Transfer in Space

Comparing the performance of within and transferred data is the first goal of the experimentation. The result of this goal will tell us whether TL can enable instances transfer between domains and time intervals without loss from the performance, or whether the transfer comes at the expense of higher error rates. Due to space constraints and also in order to make the results more readable we equally divided the domain transfer results of the Tukutuku subsets into two figures: Figure 7 and Figure 8. The time interval transfer results of Cocomo81 and Nasa93 are provided in Figure 10 and Figure 11, respectively.

Figure 7 shows a uniformity of results. The tie values are very high for 5 out of 8 companies (tuku1, tuku4-to-7), which means that transferred data performance is as good as within data performance. For 1 company, tuku8, within and transferred data performance depends on the error measure: According to all error measures except MMER the within and transferred data performances are very close, whereas for MMRE the within data performance appears to be better. For 2 companies out of 8 (tuku2 and tuku3), the within data performance is dominantly better with a win value of 20. Remember from §3.3.2 that TEAK performs 20 times LOOCV, hence the total of win, tie and loss values for each data set subject to each error measure amounts to 20.

Figure 8 shows the within and transferred data performance comparison for the error measures of MMER, MBRE, MIBRE and SA. The reading of Figure 8 is exactly the same as of Figure 7, i.e. it shows the win, tie, and loss values according to 4 error measures. The general pattern we have observed from 4 error measures in Figure 7 are also visible in Figure 8. In relation to the error measures MBRE, MIBRE and SA, in 6 data sets (tuku1, tuku2, tuku4-to-7) transferred and within data performances are the same. According to the MMER, within performance is better for 2 data sets: tuku3 and tuku8.

The aforementioned results are parallel to the prior results reported by Kocaguneli et al. [Kocaguneli and Menzies, 2011b], where they have used 21 public data sets and questioned merely the TL between domains [Boetticher et al., 2007]. A summary of their results on 21 public data sets are provided in Figure 9. As can be seen in Figure 9, Kocaguneli et al. use 4 error measures and identify only 2 data sets (gray highlighted rows) for which within data performance is worse than that of transferred data. For $21-2 = 19$ data sets, transferred data performance is shown to be statistically significantly the same as that of within.

Dataset	MAR		
	Win	**Tie**	**Loss**
tuku1	8	12	0
tuku2	20	0	0
tuku3	20	0	0
tuku4	2	18	0
tuku5	0	20	0
tuku6	8	12	0
tuku7	2	18	0
tuku8	1	19	0

Dataset	MMRE		
	Win	**Tie**	**Loss**
tuku1	9	11	0
tuku2	20	0	0
tuku3	20	0	0
tuku4	2	18	0
tuku5	0	19	1
tuku6	5	15	0
tuku7	3	17	0
tuku8	0	20	0

Dataset	MdMRE		
	Win	**Tie**	**Loss**
tuku1	8	12	0
tuku2	20	0	0
tuku3	20	0	0
tuku4	2	18	0
tuku5	0	19	1
tuku6	5	15	0
tuku7	3	17	0
tuku8	0	20	0

Dataset	Pred(25)		
	Win	**Tie**	**Loss**
tuku1	6	10	4
tuku2	18	0	2
tuku3	20	0	0
tuku4	2	18	0
tuku5	1	19	0
tuku6	5	15	0
tuku7	3	17	0
tuku8	0	20	0

Fig. 7: Performance comparison of within vs transferred data w.r.t. 4 of 8 different performance measures: MAR, MMRE, MdMRE, Pred(25). Win, tie, loss values are w.r.t. within data.

Dataset	MMER		
	Win	**Tie**	**Loss**
tuku1	0	20	0
tuku2	0	20	0
tuku3	20	0	0
tuku4	0	20	0
tuku5	6	14	0
tuku6	8	12	0
tuku7	6	14	0
tuku8	14	6	0

Dataset	MBRE		
	Win	**Tie**	**Loss**
tuku1	6	14	0
tuku2	20	0	0
tuku3	20	0	0
tuku4	1	19	0
tuku5	0	20	0
tuku6	6	14	0
tuku7	5	15	0
tuku8	2	18	0

Dataset	MIBRE		
	Win	**Tie**	**Loss**
tuku1	2	18	0
tuku2	17	3	0
tuku3	20	0	0
tuku4	1	19	0
tuku5	0	20	0
tuku6	6	14	0
tuku7	4	16	0
tuku8	3	17	0

Dataset	SA		
	Win	**Tie**	**Loss**
tuku1	8	12	0
tuku2	20	0	0
tuku3	20	0	0
tuku4	2	18	0
tuku5	0	20	0
tuku6	8	12	0
tuku7	2	18	0
tuku8	1	19	0

Fig. 8: Performance comparison of within vs transferred data w.r.t. 4 of 8 different performance measures: MMER, MBRE, MIBRE, SA. Win, tie, loss values are w.r.t. within data.

Dataset	MAR			MMRE			MdMRE			Pred(30)		
	Win	Tie	Loss	Win	Tie	Loss	Win	Tie	Loss	Win	Tie	Loss
cocomo81e	0	20	0	0	16	4	4	16	0	4	16	0
cocomo81o	0	20	0	2	18	0	2	18	0	2	18	0
cocomo81s	18	2	0	15	5	0	15	5	0	13	5	2
nasa93_center_1	0	20	0	0	20	0	0	20	0	0	20	0
nasa93_center_2	4	16	0	2	18	0	2	18	0	2	18	0
nasa93_center_5	0	20	0	0	12	8	8	12	0	8	11	1
desharnaisL1	11	9	0	9	11	0	9	11	0	9	11	0
desharnaisL2	0	20	0	0	20	0	0	20	0	0	20	0
desharnaisL3	0	20	0	2	18	0	2	18	0	2	18	0
finnishAppType1	0	20	0	0	20	0	0	20	0	0	20	0
finnishAppType2345	0	20	0	0	17	3	0	17	3	0	17	3
kemererHardware1	0	0	20	0	0	20	0	0	20	0	0	20
kemererHardware23456	0	20	0	0	20	0	0	20	0	0	20	0
maxwellAppType1	6	14	0	1	19	0	1	19	0	0	19	1
maxwellAppType2	0	18	2	0	19	1	0	19	1	0	19	1
maxwellAppType3	0	20	0	1	19	0	1	19	0	1	19	0
maxwellHardware2	0	20	0	0	20	0	0	20	0	0	20	0
maxwellHardware3	0	20	0	0	20	0	0	20	0	0	20	0
maxwellHardware5	0	20	0	0	20	0	0	20	0	0	20	0
maxwellSource1	6	14	0	1	19	0	1	19	0	1	19	0
maxwellSource2	0	20	0	0	20	0	0	20	0	0	20	0

Fig. 9: Summary of prior TEAK results [Kocaguneli and Menzies, 2011b] on 21 public data sets, within vs transferred data (i.e. win, tie, loss values are w.r.t. within data.). For 19 data sets, transferred and within data performances are the same (note high tie values). For only 2 data sets (highlighted) within data performance is better.

4.2 Transfer in Time

Figure 10 and Figure 11 show the performance results of TL in time intervals for Cocomo81 and Nasa93. For Cocomo81, two within sources are defined: 1) projects developed from 1960 to 1975 (called as coc-60-75) and 2) projects developed from 1976 onwards (called as coc-76-rest). Similarly, the subsets of Nasa93 are: 1) projects from 1970 to 1979 (called as nasa-70-79) and 2) projects from 1980 onwards (called as nasa-80-rest). In both Figure 10 and Figure 11, the tie values are quite high with the smallest tie value of 16. Note that in none of the two figures there is a highlighted row, which means that in none of the time interval instance transfer experiments was there a case where we failed to transfer instances between different time intervals. The implications of within and transferred data experiments through instance transfer in time intervals are important for practitioners. TL results on Cocomo81 and Nasa93 subsets show that TL methods, may help companies use aged data sets by identifying which instances are still relevant, hence can be transfered to contemporary estimation tasks. Figure 10 and Figure 11 fundamentally show that decades of time difference can be crossed with the help of instance transfer.

Dataset	MAR		
	Win	**Tie**	**Loss**
coc-60-75	0	20	0
coc-76-rest	0	20	0
Dataset	**MMRE**		
	Win	**Tie**	**Loss**
coc-60-75	0	19	1
coc-76-rest	0	20	0
Dataset	**MdMRE**		
	Win	**Tie**	**Loss**
coc-60-75	0	19	1
coc-76-rest	0	20	0
Dataset	**Pred(25)**		
	Win	**Tie**	**Loss**
coc-60-75	0	19	1
coc-76-rest	0	20	0
Dataset	**MMER**		
	Win	**Tie**	**Loss**
coc-60-75	0	20	0
coc-76-rest	0	20	0
Dataset	**MBRE**		
	Win	**Tie**	**Loss**
coc-60-75	1	19	0
coc-76-rest	0	20	0
Dataset	**MIBRE**		
	Win	**Tie**	**Loss**
coc-60-75	0	19	1
coc-76-rest	0	20	0
Dataset	**SA**		
	Win	**Tie**	**Loss**
coc-60-75	0	20	0
coc-76-rest	0	20	0

Fig. 10: Performance comparison of Cocomo81 subsets for transfer learning in time. Win, tie, loss values are w.r.t. within data.

Dataset	MAR		
	Win	**Tie**	**Loss**
nasa-70-79	0	20	0
nasa-80-rest	4	16	0
Dataset	**MMRE**		
	Win	**Tie**	**Loss**
nasa-70-79	0	20	0
nasa-80-rest	4	16	0
Dataset	**MdMRE**		
	Win	**Tie**	**Loss**
nasa-70-79	0	20	0
nasa-80-rest	4	16	0
Dataset	**Pred(25)**		
	Win	**Tie**	**Loss**
nasa-70-79	0	20	0
nasa-80-rest	4	16	0
Dataset	**MMER**		
	Win	**Tie**	**Loss**
nasa-70-79	4	16	0
nasa-80-rest	0	20	0
Dataset	**MBRE**		
	Win	**Tie**	**Loss**
nasa-70-79	2	18	0
nasa-80-rest	2	18	0
Dataset	**MIBRE**		
	Win	**Tie**	**Loss**
nasa-70-79	2	18	0
nasa-80-rest	2	18	0
Dataset	**SA**		
	Win	**Tie**	**Loss**
nasa-70-79	0	20	0
nasa-80-rest	4	16	0

Fig. 11: Performance comparison of Nasa93 subsets for transfer learning in time. Win, tie, loss values are w.r.t. within data.

The results of the TL research reported herein on proprietary data sets combined with the prior results of public data sets [Kocaguneli and Menzies, 2011b] give us a broader picture of the transferred data performance. By using instance transfer methods, such as TEAK between domains, we have:

- 5 out of 8 proprietary data sets (results of this study);
- 19 out of 21 public data sets (results of prior study);

where the performance difference between within and transferred data is not statistically significant. This shows us that from a total of $21 + 8 = 29$ public and proprietary data sets, transferred data performs as well as within data for $5 + 19 = 24$ cases. Also, as for the TL between time intervals, we have 2 data sets subject to 8 error measures ($2 \times 8 = 16$ cases), where transferred data performance is always the same as within data performance.

4.3 Inspecting Selection Tendencies

The second goal of our experimentation is to observe the selection tendencies of the test instances. Figure 12 shows what percentage of instances are selected from within data sources (diagonal cells) as well as the amount of the transferred data (off diagonal cells) in TL experiments between domains. The first column of Figure 12 shows the within data sources (8 different companies) as well as their sizes in parenthesis. The second column shows the number of analogies retrieved from GAC2 on average over 20 runs. For each row, the columns *tuku1-to-8* show how the number of analogies in the second column is distributed to each data source. The values outside the parenthesis in each cell of columns *tuku1-to-8* are the number of analogies selected from that data source; the percentage value of that number w.r.t. the second column is given inside the parenthesis. Figure 13 and Figure 14 show the selection tendency of test instances for Cocomo81 and Nasa93 time interval TL experiments. These figures are structured in the same manner as Figure 12.

Dataset	**# of Analogies**	**tuku1**	**tuku2**	**tuku3**	**tuku4**	**tuku5**	**tuku6**	**tuku7**	**tuku8**
tuku1 (14)	5.8	0.5 (3.7)	1.0 (4.8)	0.1 (0.8)	0.1 (2.3)	0.4 (2.8)	0.4 (4.9)	1.7 (5.4)	1.6 (9.1)
tuku2 (20)	11.0	1.1 (8.0)	1.6 (8.2)	0.3 (2.0)	0.8 (12.9)	0.7 (5.0)	0.7 (8.3)	2.7 (8.7)	3.1 (17.2)
tuku3 (15)	7.3	0.9 (6.2)	1.3 (6.3)	0.4 (2.9)	0.2 (4.1)	0.8 (6.4)	0.2 (2.9)	1.5 (4.9)	1.9 (10.5)
tuku4 (6)	6.7	0.6 (4.4)	1.5 (7.6)	0.2 (1.5)	0.3 (5.0)	0.2 (1.7)	0.7 (8.9)	1.2 (3.8)	2.0 (10.9)
tuku5 (13)	9.3	2.4 (17.2)	1.4 (7.2)	0.3 (2.1)	0.5 (8.5)	0.5 (4.2)	0.4 (5.4)	1.3 (4.0)	2.4 (13.1)
tuku6 (8)	8.9	0.7 (4.7)	1.6 (7.8)	0.2 (1.5)	0.7 (12.4)	0.7 (5.5)	0.7 (8.4)	1.8 (5.7)	2.6 (14.3)
tuku7 (31)	7.8	1.2 (8.3)	1.3 (6.4)	0.3 (2.3)	0.5 (8.4)	0.6 (4.5)	0.1 (1.5)	1.7 (5.5)	2.1 (11.6)
tuku8 (18)	6.9	1.1 (7.9)	1.1 (5.7)	0.3 (2.3)	0.3 (4.5)	0.7 (5.5)	0.3 (3.3)	1.3 (4.3)	1.8 (9.7)

Fig. 12: The amount of within and transferred data selected. The first column is the subset names and their sizes in parenthesis. The second column is the average number of retrieved instances in GAC2. The following columns show the number of analogies from each particular subset, in parenthesis the percentage values of these numbers w.r.t. the second column are given.

Dataset	# of Analogies	coc-60-75	coc-76-rest
coc-60-75 (20)	3.6	0.8 (4.1)	2.8 (6.4)
coc-76-rest (43)	4.5	1.3 (6.6)	3.2 (7.4)

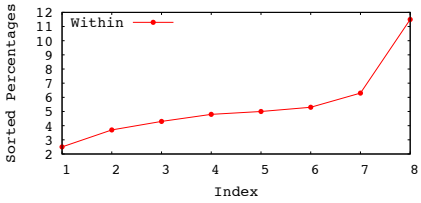
Fig. 13: The amount of within and transferred data from subsets of Cocomo81.

Dataset	# of Analogies	nasa-70-79	nasa-80-rest
nasa-70-79 (39)	7.3	2.6 (6.6)	4.7 (8.7)
nasa938089 (54)	9.5	3.2 (8.2)	6.3 (11.7)

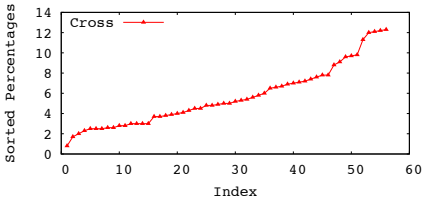
Fig. 14: The amount within and transferred data selected from the subsets of Nasa93.

The selection tendency values of Tukutuku, Cocomo81 and Nasa93 subsets provide us with suggestions regarding how much data a test instance uses from within and transferred data during the domain and time interval transfer of training instances. From the selection tendency figures, we can identify two findings:
Finding #1: See the second columns of Figure 12, Figure 13 and Figure 14 that only a very small portion of all the available data is transferred as useful analogies. This finding points to the importance of: a) instance transfer, a.k.a filtering; b) TL methods like TEAK that are capable of transferring relevant analogies between domains and time intervals. The low number of instances selected are also supported by relevant literature: Chang's prototype generators [Chang, 1974] replaced training sets of size $T = (514, 150, 66)$ with prototypes of size $N = (34, 14, 6)$ (respectively). That is, prototypes may be as few as $\frac{N}{T} = (7, 9, 9)\%$ of the original data. Note that these values are close to how many instances were retrieved in the above results.
Finding #2: When we compare the diagonal and off-diagonal percentages, we see that the values are very close. The second finding bears importance regarding whether or not the TL is worthy of investigation. We see that the amount of data transfered from other domains or time intervals is as much as the instances used from the within data sources. In other words, there is a considerable amount of data that is transfered as relevant across time and space. In addition (recall from the previous subsection) the high amount of transfered data does not come at the expense of higher error rates. This shows us that discarding aged data sets because they are old is a misguided practice. There are relevant instances in old data sets that are relevant to current practices.

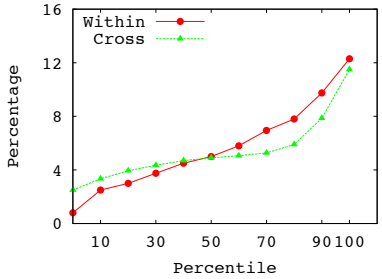
To better observe how close within and transferred data percentages are, we plot the percentage values of within and transferred data sources of Tukutuku subsets in Figure 15(a) and Figure 15(b), respectively. Figure 15(a) and Figure 15(b) are basically the plots of diagonal and off-diagonal percentage values of Figure 12. We see from



(a) Within Percentages



(b) Transfer Data Percentages



(c) Within and Tranfer Data Percentiles

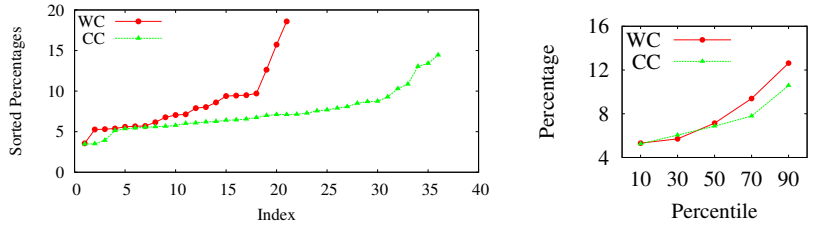
Fig. 15: Percentages of instances (a.k.a. analogies) from (a) within and (b) transferred data. The percentage values come from Figure 12. Within company percentages are the gray-highlighted diagonal cells, whereas transferred data percentages are the remaining off-diagonal cells. The percentile graph (c) shows the percentiles of (a) and (b).

Figure 15(a) and Figure 15(b) that the maximum and minimum percentages of within and transferred data selection are very close. To align these percentages, we took the percentile values from 0^{th} percentile to 100^{th} percentile with increments of 10. The resulting percentiles are shown in Figure 15(c). See in Figure 15(c) that within and transferred data have similar percentile values, i.e. the selection tendencies are very close to one another. Note that percentage and percentile plots are unnecessary for Figure 13 and Figure 14, since the closeness of within and transfer data percentages of Cocomo81 and Nasa93 subsets can easily be verified with manual inspection: For Cocomo81 subsets the biggest percentage difference is $6.6 - 4.1 = 2.5\%$; for Nasa93 subsets it is $11.7 - 8.2 = 3.5\%$.

Figure 16 is taken from Kocaguneli et al.'s selection tendency experiments [Kocaguneli and Menzies, 2011b]. In the performance experiments, we have seen the similarity between the results of this research and that of Kocaguneli et al. in terms of performance. Comparison of Figure 16 to Figure 15 shows that the similarity of results are also valid in terms of the selection tendencies. See in particular the percentile values of Figure 15 and Figure 16 that TL between domains of proprietary data sets and public data sets have similar selection tendencies.

5 Threats to Validity

Internal validity questions the extent to which the association between dependent and independent variables holds [Alpaydin, 2010]. To observe this relationship SEE studies



(a) Percentages (b) Percentiles

Fig. 16: Percentages and percentiles of instances from within and transferred data as given by Kocaguneli et al. [Kocaguneli and Menzies, 2011b]. Note the similarity of the percentile plot of this figure to that of Figure 15.

make use of dividing data sets into test and train data. This division is performed in accordance with various sampling methods such as LOOCV or any N-Way cross validation. However, selecting any particular sampling method is inherently an internal validity threat [Hastie et al., 2008] as different sampling methods are expected to have different biases and variance values. In this research LOOCV is employed, whose alternative is any N-Way cross-validation. In a N-Way cross-validation, data is randomly divided into B bins and each bin is tested on a model learned from the combination of other bins (typical values for B are 3 or 10). Elsewhere [Kocaguneli and Menzies, 2012], we show that there is very little difference in the bias and variance values generated for LOOCV and N-way cross-validation. Since two testing strategies have similar bias-variance characteristics for effort datasets, we opted for LOOCV due to the fact that LOOCV is a deterministic procedure that can be exactly repeated by any other researcher with access to a particular data set.

External validity deals with the generality of the results, i.e. whether the results can be generalized outside the specifications of a study [Milicic and Wohlin, 2004]. This study challenges the validity of prior results of 21 within and transfer data source pairs [Kocaguneli and Menzies, 2011b] on proprietary data sets coming from 8 Web development companies. The results of this study confirm prior findings, i.e. a total of $21 + 8 = 29$ data sets agree on the reported conclusions, which is way more extensive than other TL studies. Among 10 studies investigated by Kitchenham et al. in [Kitchenham et al., 2007], 9 of them used single within-transfer dataset pairs, and 1 study used 6 pairs. Therefore, concerning external validity, this research bears higher validity than other similar studies. However, it is important to note that none of the data sets employed here represents a random sample; therefore the results presented herein are much more likely to scale to those companies that manage projects similar to those detailed here.

Construct validity (a.k.a. face validity) observes whether a study measures what it actually intends to measure [Robson, 2002]. Previous studies have concerned themselves with the construct validity of different performance measures for effort estimation (e.g. [Stensrud et al., 2002]). The intention of this paper is neither providing evidence for a particular error measure nor biasing our conclusions due to selection of one particular error measure. Instead, we used 8 different performance measures (which covers a big majority of all the previously used error measures in SEE) aided with win-tie-loss statistics so that our results are able to be benchmarked against other studies.

6 Future Research

Based on the majority of the companies in the domain TL experiments (depending on the error measure, 5 or 6 companies out of 8) the transferred data performance is the same as the within data performance. In terms of time interval TL, in *all* of the cases, within and transferred data performances were statistically the same. This shows us that TL through methods like TEAK can help transfer instances between domains and time intervals so that transferred data can perform as well as within data.

That said, an interesting fact is that error measures can result in different conclusions. For example, see in Figure 7 and Figure 8 that transferred data performance for the company tuku8 depends on error measures. This disagreement may cause different companies to make different conclusions depending on the particular error measures they are using. Hence as part of the future work we intend to investigate error measure studies like that of Shepperd et al. [Shepperd and Macdonell, 2011], as they bear a significant importance for SEE as well as for the recommendations to practitioner audiences.

Another area of future work would be the exploration of other kinds of transfer learning. For example, this paper has employed *instance-based transfer* where the instances share the same ontology (names of column headers). A more challenging task would be to explore instance-based transfer to take data from different projects collected using different ontologies. For example, [Pan and Yang, 2010] described experiments in image processing where data is collected using 2D and 3D sensors, then both projected and clustered within a new synthesized set of dimensions. The key to making that work is to find some (possible small) number of common reference points to enable the mapping of the raw dimensions onto the synthesized dimensions. Our own experiments in that direction are encouraging, but very preliminary. If other effort estimation researchers want to explore this issue, then we offer them the *Rosetta Stone challenge*:

- The COCOMO Rosetta Stone was create by Reifer et al. [Reifer et al., 1999] to translate between COCOMO-81 effort estimation model and the revised COCOMO-2000 model (the new model had some new parameters and deleting some old ones).
- The *Rosetta Stone Challenge* is to automatically recreate that manually derived translator from the COCOMO-81 to COCOMO-2000 ontology (or, alternatively, show that some better translator exists).
- Note that COCOMO-81 and COCOMO-2000 data sets are available at the PROMISE web site promisedata.googlecode.com.

7 Conclusion

When projects lack sufficient local data to make predictions, they can try to transfer information from other projects. In the field of software engineering, transfer learning has been shown to be effective for defect prediction. This paper checked if it was possible to build transfer learners for effort estimation. We explored the following research questions.

RQ1: Is transfer learning possible outside the area of defect prediction?

We find that an analogy-based learner called TEAK that clusters together similar projects can be used for transfer learning. Such a learner has the following properties that make it useful for transfer learning:

1. Regardless of the source of the data, TEAK groups together similar data from different stratifications.
2. TEAK makes its conclusions using the local neighborhood, regardless of what stratifications generated that neighborhood.

RQ2: Is transfer learning effective for effort estimation?

We compared estimates learned from within a single stratification to those transferred across different stratifications. In the majority case, the cross-stratification estimates transferred by TEAK do as well as the within-stratification estimates.

RQ3: How useful are manual divisions of the data?

The selection tendency results of this research show that test instances select equal amounts of instances from within and transferred data sources; i.e. TEAK found no added value in restricting reasoning to just within a delphi localization. We therefore advise that delphi localization should be used with great care, perhaps only after checking if automatic clustering methods can generate better estimates than clustering after delphi localization.

RQ4: Does transfer learning for effort estimation work across time as well as space?

This paper has tested a conjecture from Storkey [Storkey, 2009] that what is called "data set shift" and "cross-company learning" are both instances of TL. To explore that conjecture, this paper successfully applied the same transfer learner *without modification* to traditional cross-company learning problems as well as data set shift problems. Hence, we would suggest that it is useful to combine research methods for two previously unrelated research threads in SE: Turhan's "data set shift" and Kitchenham et al.'s "cross-company learning".

Note also that, to the best of our knowledge, this is the first report in the effort estimation literature of a successful transfer learning application that can effectively transfer data across time and space.

RQ5: Are repositories of software project data valuable to industrial companies?

The above shows that, for the particular purpose of effort estimation, such repositories let us predict properties in new projects. Hence, the conclusion of this article is that for that purpose, repositories of software project data are valuable to industry.

RQ6: Does the history of software engineering have relevant lessons for today?

Our results show that it is misguided to think that:

- The data of another organization cannot be used for local solutions.
- Old data of an organization is irrelevant to current context. This is a very important point. The success of our transfer learners in moving data means that we should not always discard the hard-won lessons of the past. Our history has lessons that are still relevant today.

References

[Alpaydin, 2010] Alpaydin, E. (2010). *Introduction to Machine Learning, 2nd edn.* MIT Press.

[Arnold et al., 2007] Arnold, A., Nallapati, R., and Cohen, W. (2007). A comparative study of methods for transductive transfer learning. In *ICDM'07: Seventh IEEE International Conference on Data Mining Workshops*, pages 77 –82.

[Bettenburg et al., 2012] Bettenburg, N., Nagappan, M., and Hassan, A. E. (2012). Think locally, act globally: Improving defect and effort prediction models. In *MSR'12*.

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

[Boetticher et al., 2007] Boetticher, G., Menzies, T., and Ostrand, T. (2007). PROMISE repository of empirical software engineering data.

[Chang, 1974] Chang, C.-l. (1974). Finding Prototypes for Nearest Classifiers. *IEEE Transactions on Computer*, C3(11).

[Corazza et al., 2010] Corazza, A., Di Martino, S., Ferrucci, F., Gravino, C., Sarro, F., and Mendes, E. (2010). How effective is tabu search to configure support vector regression for effort estimation? In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*.

[D. Rodriguez and Harrison, 2012] D. Rodriguez, I. H. and Harrison, R. (2012). On software engineering repositories and their open problems. In *Proceedings RAISE'12*.

[Dai et al., 2007] Dai, W., Xue, G.-R., Yang, Qiang, and Yong, Y. (2007). Transferring naive bayes classifiers for text classification. In *AAAI'07: Proceedings of the 22nd national conference on Artificial intelligence*, pages 540–545.

[Foss et al., 2003] Foss, T., Stensrud, E., Kitchenham, B., and Myrtveit, I. (2003). A simulation study of the model evaluation criterion mmre. *IEEE Trans. Softw. Eng.*, 29(11):985–995.

[Foster et al., 2010] Foster, G., Goutte, C., and Kuhn, R. (2010). Discriminative instance weighting for domain adaptation in statistical machine translation. In *EMNLP '10: Conference on Empirical Methods in Natural Language Processing*, pages 451–459.

[Gao et al., 2008] Gao, J., Fan, W., Jiang, J., and Han, J. (2008). Knowledge transfer via multiple model local structure mapping. In *In International Conference on Knowledge Discovery and Data Mining, Las Vegas, NV*.

[Harman et al., 2012] Harman, M., Jia, Y., and Zhang, Y. (2012). App store mining and analysis: Msr for app stores. In *MSR*, pages 108–111.

[Hastie et al., 2008] Hastie, T., Tibshirani, R., and Friedman, J. (2008). *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 edition.

[Hayes et al., 2006] Hayes, J. H., Dekhtyar, A., and Sundaram, S. K. (2006). Advancing candidate link generation for requirements tracing: The study of methods. *IEEE Trans. Software Eng*, 32(1):4–19.

[He et al., 2012] He, Z., Shu, F., Yang, Y., Li, M., and Wang, Q. (2012). An investigation on the feasibility of cross-project defect prediction. *Automated Software Engineering*, 19:167–199.

[Hihn and Habib-agahi, 1991] Hihn, J. and Habib-agahi, H. (1991). Cost estimation of software intensive projects: a survey of current practices. In *Software Engineering, 1991. Proceedings., 13th International Conference on*, pages 276 –287.

[Hindle, 2012] Hindle, A. (2012). Green mining: A methodology of relating software change to power consumption. In *Proceedings, MSR'12*.

[Huang et al., 2007] Huang, J., Smola, A., Gretton, A., Borgwardt, K., and Scholkopf, B. (2007). Correcting sample selection bias by unlabeled data. In *Proceedings of the 19th Annual Conference on Neural Information Processing Systems*, pages 601–608.

[Jiang et al., 2008] Jiang, Y., Cukic, B., Menzies, T., and Bartlow, N. (2008). Comparing design and code metrics for software quality prediction. In *Proceedings, PROMISE 2008*, pages 11–18.

[Kadoda et al., 2000] Kadoda, G., Cartwright, M., and Shepperd, M. (2000). On configuring a case-based reasoning software project prediction system. *UK CBR Workshop, Cambridge, UK*, pages 1–10.

[Keung, 2008] Keung, J. (2008). Empirical evaluation of analogy-x for software cost estimation. In *ESEM '08: Proceedings of the Second International Symposium on Empirical Software Engineering and Measurement*, pages 294–296, New York, NY, USA. ACM.

[Keung et al., 2012] Keung, J., Kocaguneli, E., and Menzies, T. (2012). Finding conclusion stability for selecting the best effort predictor in software effort estimation. *Automated Software Engineering*, pages 1–25. 10.1007/s10515-012-0108-5.

[Kitchenham et al., 2007] Kitchenham, B. A., Mendes, E., and Travassos, G. H. (2007). Cross versus within-company cost estimation studies: A systematic review. *IEEE Trans. Softw. Eng.*, 33(5):316–329.

[Kocaguneli et al., 2010] Kocaguneli, E., Gay, G., Yang, Y., Menzies, T., and Keung, J. (2010). When to use data from other projects for effort estimation. In *ASE '10: Proceedings of the International Conference on Automated Software Engineering (short paper)*, New York, NY, USA.

[Kocaguneli and Menzies, 2011a] Kocaguneli, E. and Menzies, T. (2011a). How to find relevant data for effort estimation? In *Proceedings ESEM'11*.

[Kocaguneli and Menzies, 2011b] Kocaguneli, E. and Menzies, T. (2011b). How to find relevant data for effort estimation. In *ESEM'11: International Symposium on Empirical Software Engineering and Measurement*.

[Kocaguneli and Menzies, 2012] Kocaguneli, E. and Menzies, T. (2012). Software effort models should be assessed via leave-one-out validation. *Under Review*.

[Kocaguneli et al., 2012] Kocaguneli, E., Menzies, T., Bener, A., and Keung, J. W. (2012). Exploiting the essential assumptions of analogy-based effort estimation. *IEEE Trans. Softw. Eng.*, 38(2):425 –438.

[Lee et al., 2007] Lee, S.-I., Chatalbashev, V., Vickrey, D., and Koller, D. (2007). Learning a meta-level prior for feature relevance from multiple related tasks. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 489–496.

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

[Lokan and Mendes, 2009a] Lokan, C. and Mendes, E. (2009a). Applying moving windows to software effort estimation. In *ESEM'09: Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 111–122.

[Lokan and Mendes, 2009b] Lokan, C. and Mendes, E. (2009b). Using chronological splitting to compare cross- and single-company effort models: further investigation. In *Proceedings of the Thirty-Second Australasian Conference on Computer Science - Volume 91*, ACSC '09, pages 47–54.

[Ma et al., 2012] Ma, Y., Luo, G., Zeng, X., and Chen, A. (2012). Transfer learning for cross-company software defect prediction. *Information and Software Technology*, 54(3):248 – 256.

[Mendes and Mosley, 2008] Mendes, E. and Mosley, N. (2008). Bayesian network models for web effort prediction: A comparative study. *IEEE Trans. Softw. Eng.*, 34:723–737.

[Mendes et al., 2005] Mendes, E., Mosley, N., and Counsell, S. (2005). Investigating web size metrics for early web cost estimation. *The Journal of Systems and Software*, 77:157–172.

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

[Menzies et al., 2012] Menzies, T., Butcher, A., Cok, D., Marcus, A., Layman, L., Shull, F., Turhan, B., and Zimmermann, T. (2012). Local vs. global lessons for defect prediction and effort estimation. *IEEE Transactions on Software Engineering*, page 1.

[Menzies et al., 2011] Menzies, T., Butcher, A., Marcus, A., Zimmermann, T., and Cok, D. (2011). Local vs global models for effort estimation and defect prediction. In *IEEE ASE'11*. Available from `http://menzies.us/pdf/11ase.pdf`.

[Menzies et al., 2007] Menzies, T., Greenwald, J., and Frank, A. (2007). Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*. Available from `http://menzies.us/pdf/06learnPredict.pdf`.

[Mihalkova et al., 2007] Mihalkova, L., Huynh, T., and Mooney, R. J. (2007). Mapping and revising markov logic networks for transfer learning. In *AAAI'07: Proceedings of the 22nd national conference on Artificial intelligence*, pages 608–614.

[Milicic and Wohlin, 2004] Milicic, D. and Wohlin, C. (2004). Distribution patterns of effort estimations. In *Euromicro Conference series on Software Engineering and Advanced Applications*, pages 422–429.

[Pan and Yang, 2010] Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345 –1359.

[Petersen and Wohlin, 2009] Petersen, K. and Wohlin, C. (2009). Context in industrial software engineering research. In *Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on*, pages 401 –404.

[Posnett et al., 2011] Posnett, D., Filkov, V., and Devanbu, P. (2011). Ecological inference in empirical software engineering. In *Proceedings of ASE'11*.

[Reifer et al., 1999] Reifer, D., Boehm, B. W., and Chulani, S. (1999). The Rosetta Stone: Making COCOMO 81 Estimates Work with COCOMO II. *Crosstalk. The Journal of Defense Software Engineering.*, pages 11–15.

[Robson, 2002] Robson, C. (2002). Real world research: a resource for social scientists and practitioner-researchers. *Blackwell Publisher Ltd.*

[Shepperd and Macdonell, 2011] Shepperd, M. and Macdonell, S. (2011). Evaluating Prediction Systems in Software Project Estimation. *Information and Software Technology*, preprints:1–21.

[Shepperd and MacDonell, 2012] Shepperd, M. and MacDonell, S. (2012). Evaluating prediction systems in software project estimation. *Information and Software Technology (preprint)*.

[Shepperd and Schofield, 1997] Shepperd, M. and Schofield, C. (1997). Estimating software project effort using analogies. *IEEE Trans. Softw. Eng.*, 23(11):736–743.

[Stensrud et al., 2002] Stensrud, E., Foss, T., Kitchenham, B., and Myrtveit, I. (2002). An empirical validation of the relationship between the magnitude of relative error and project size. In *Proceedings of the Eighth IEEE Symposium on Software Metrics*, pages 3 – 12.

[Storkey, 2009] Storkey, A. (2009). When training and test sets are different: characterizing learning transfer. *Dataset Shift in Machine Learning (J. Candela, M. Sugiyama, A. Schwaighofer and N. Lawrence, eds.). MIT Press, Cambridge, MA*, pages 3–28.

[Turhan, 2012] Turhan, B. (2012). On the dataset shift problem in software engineering prediction models. *Empirical Software Engineering*, 17:62–74.

[Turhan et al., 2009] Turhan, B., Menzies, T., Bener, A., and Di Stefano, J. (2009). On the relative value of cross-company and within-company data for defect prediction. *Empirical Software Engineering*, 14(5):540–578.

[Wu and Dietterich, 2004] Wu, P. and Dietterich, T. G. (2004). Improving svm accuracy by training on auxiliary data sources. In *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, pages 110–, New York, NY, USA. ACM.

[Yang et al., 2011] Yang, Y., Xie, L., He, Z., Li, Q., Nguyen, V., Boehm, B. W., and Valerdi, R. (2011). Local bias and its impacts on the performance of parametric estimation models. In *PROMISE*.

[Zhang and Sheng, 2004] Zhang, H. and Sheng, S. (2004). Learning weighted naive bayes with accurate ranking. In *ICDM '04. Fourth IEEE International Conference on Data Mining*, pages 567 – 570.

[Zhang et al., 2007] Zhang, X., Dai, W., Xue, G.-R., and Yu, Y. (2007). Adaptive email spam filtering based on information theory. In *Web Information Systems Engineering WISE 2007*, volume 4831 of *Lecture Notes in Computer Science*, pages 159–170. Springer Berlin / Heidelberg.

[Zimmermann et al., 2009] Zimmermann, T., Nagappan, N., Gall, H., Giger, E., and Murphy, B. (2009). Cross-project defect prediction: A large scale experiment on data vs. domain vs. process. *ESEC/FSE*, pages 91–100.