Filter-INC: Handling Effort-Inconsistency in Software Effort Estimation Datasets

Passakorn Phannachitta*, Jacky Keung†, Kwabena Ebo Bennin†, Akito Monden‡, and Kenichi Matsumoto*
*Graduate School of Information Science, Nara Institute of Science and Technology, Japan
†Department of Computer Science, City University of Hong Kong, China
‡Graduate School of Natural Science and Technology, Okayama University, Japan
Email: {phannachitta-p,matumoto}@is.naist.jp, {Jacky.Keung,kebennin2-c}@cityu.edu.hk, monden@okayama-u.ac.jp

***Abstract*—Effort-inconsistency is a situation where historical software project data used for software effort estimation (SEE) are contaminated by many project cases with similar characteristics but are completed with significantly different amount of effort. Using these data for SEE generally produces inaccurate results; however, an effective technique for its handling is yet made to be available. This study approaches the problem differently from common solutions, where available techniques typically attempt to remove every project case they have detected as outliers. Instead, we hypothesize that data inconsistency is caused by only a few deviant project cases and any attempt to remove those other cases will result in reduced accuracy, largely due to loss of useful information and data diversity. *Filter-INC* (short for Filtering technique for handling effort-INConsistency in SEE datasets) implements the hypothesis to decide whether a project case being detected by any existing technique should be subject to removal. The evaluation is carried out by comparing the performance of 2 filtering techniques between before and after having *Filter-INC* applied. The results produced from 8 real-world datasets together with 3 machine-learning models, and evaluated by 4 performance measures show a significant accuracy improvement at the confident interval of 95%. Based on the results, we recommend our proposed hypothesis as an important instrument to design a data preprocessing technique for handling effort-inconsistency in SEE datasets, definitely an important step forward in preprocessing data for a more accurate SEE model.**

***Index Terms*—Effort-inconsistency, Data preprocessing, Software effort estimation, Empirical software engineering**

I. Introduction

Model-based software effort estimation (SEE) methodology adopts machine-learning models to generate the expected amount of development effort required by a new software project. Utilizing this methodology, data quality is one of the most important instrument to obtain high accuracy. In this study, we investigate a problem where historical project cases with similar project features were completed with a diversified amount of effort. Past studies such as Le-do et al. [17] called this situation as effort-inconsistency. This situation is commonly encountered in real-world datasets but is considerably difficult to handle [19]. The presence of these contaminated project cases strongly hinders the adoption of accurate estimation models because of the increase in uncertainties and discrepancies which defines the assumptions supporting any learning model. For example, it is difficult to justify and rely on information obtained from historical project cases if records of any two project cases developed using the same programming language with similar software size surprisingly have different and wide effort values.

Unfortunately, very few attempts to properly handle this undesirable situation have been made [19], clearly indicating that SEE in practice is facing difficulty in achieving desirable accuracy, regardless of how advanced the learning models are. We assume that the lack of a solid solution is mainly because available techniques more typically attempt to increase the level of effort-consistency in datasets by removing project cases that can be necessary. For example, a technique named *FISI* [17] quantifies the level of effort-inconsistency of pairs of project cases, and suggests to remove all the pairs whose level of effort-inconsistency are higher than that of the others, i.e., says the highest 10% portion. We argue the correctness for this procedure that, when observing project cases in pairs and two cases of a pair are detected as being effort-inconsistent, then, only one of the two project cases might be the deviant, whereas the other one could be consistent with the remaining cases in the dataset.

Regarding the challenge that SEE datasets are typically considered as being very sparse and lack diversity [16], any attempt to remove any project cases that are important is more likely to reduce the estimation performance. Therefore, we propose a hypothesis for better handling of effort-inconsistency that *instead of removing all the project cases considered as effort-inconsistent, an increase in performance would be obtained if only the deviant project cases can be identified and removed*. It is important to note at this point that deviant cases considered in this study is different from outliers, where isolation of project cases based on outliers is observed by means of the entire population. On the other hand, deviant cases are observed by means of inter-relationship between project cases. Subsequently, we propose *Filter-INC* (short for Filtering technique for data INConsistency handling) as an implementation of the hypothesis. It is proceeded by scrutinizing one project case at a time, finding its conflicted cases, and deciding to remove it only if effort-inconsistency is still observed between it and any other remaining cases in the dataset after temporary excluding its conflicted cases.

In evaluation, the performance of 2 existing filtering techniques named *FISI* [17] and *TEAK* [14] are adopted to compare between having *Filter-INC* applied to them and without

1530-1362/16 $31.00 © 2016 IEEE
DOI 10.1109/APSEC.2016.48

having it applied. The comparison is subject to 8 industrial datasets, 3 common SEE models, 4 performance measures, and a statistical significant test method named Brunner's test [5], recently suggested by Kitchenham [11] to be one of the most robust and reliable statistical significant test methods for empirical software engineering research studies. The results show that after *Filter-INC* is applied, the performance of both selected techniques are significantly improved. In an extended experiment where all the treatments are compared by means of ranking, we found that applying *Filter-INC* to the technique named *TEAK* is overall the highest performer of our experimental method. This allows us to conclude that our proposed hypothesis holds true and a filter technique adopted based on it, such as *Filter-INC*, should have an important role in the data preparation processes of SEE in the future.

The rest of the paper is organized as follows. Section II overviews SEE and data inconsistency. Section III explains our proposed *Filter-INC*. Section IV and V explain the experimental setup and the results, respectively. Next, Section VI discusses the findings in detail. Finally, we conclude the paper and address our future work in VII.

II. Background

A. Model-based Software Effort Estimation (SEE)

Model-based SEE methodology reuses historical software project data as references to estimate the required effort for new and upcoming projects. The modeling process is proceeded by applying machine-learning algorithms to explore the relationships between a set of descriptive variables (commonly referred to as project features) and one single target variable (the development effort). The machine-learning algorithms commonly seen in SEE literature are in 4 families: regression, tree-based models, neural networks, and analogy-based models. Of these models, a study by Keung et al. [10] successfully concluded based on statistical methods that a model named classification and regression tree (*CART*) configured with its default tree-pruning parameters is the general best model for SEE activity, and suggested to be adopted as a standard benchmark SEE model in SEE research studies.

B. The Problem of Effort-Inconsistency

Following Le-do et al. [17], the term effort-inconsistency denotes the situation concerning historical project cases whose characteristics are similar but the amount of effort spent to complete them are significantly different. Because model-based SEE typically relies its estimation processes only on historical software project data [21], quality of the data definitely affects the estimation performance of any model built. Different from conventional studies in SEE where outliers are commonly discussed [3], [4], this study observes that effort-inconsistency is also typically encountered in real-world datasets but is far less frequently discussed in the literature. In our previous study [19], we investigated effort-inconsistency and found that the difficulty in adopting an accurate SEE model increases as the number of effort-inconsistent project cases increases. Neither an efficient and reliable technique nor a standard procedure for its handling seems to have been previously established. Precisely, until now in the empirical software engineering community, little attention has been paid to the handling of this data quality issue [3], [4]. Among the existing techniques, we found 2 techniques that have their essential principle applicable for its handling. A detailed explanation of these techniques are given below:

1) FISI: Le-Do et al. [17] proposed a technique named *FISI* (short for Filtering the Inconsistent Software project data Instances) as a filtering technique to remove project cases assumed to be contaminated with effort-inconsistency from SEE datasets. Its procedure is to quantify the degree of effort-inconsistency (the authors called it Effort inconsistency degree (EID)) of project cases based on experimentations. However, despite the definition of EID being theoretically driven, the final decision to decide whether to remove a project case based on EID only relies on a less logical heuristic, i.e., to remove 10% of project cases from the dataset whose EID are of the highest values.

FISI suggests to remove a project case$_i$ from a dataset if the effort values of its k nearest neighbors are significantly different from that of the case$_i$. In steps, *FISI* configured with the parameters suggested in [17] is proceeded as follows:

1) Randomly split dataset into 10 partitions (i.e., 10 folds). Calculate the nearest neighbor lists for all the project cases by computing Euclidean distance from each project case to all the other cases located in other partitions, and ranking the project cases based on the calculated distance values.
2) Compute the relative error of effort: $r = \frac{|e_i - e'_i|}{e'_i}$, where e_i is the effort of a case$_i$ and e'_i is the effort of the nearest neighbor of the case$_i$. Note that case$_i$ and case$'_i$ are from different partitions. Set the r at the 25^{th} percentile as a threshold q.
3) Set EID$_i$ corresponding to a case$_i$ to 0.
4) For all the 3-nearest neighbor of a case$_i$, when case$_i$ and its j^{th} neighbor are nearest neighbor of each other:
$$\text{EID}_i \;=\; \textit{EID}_i + \begin{cases} \frac{1}{j} & \textit{if} \;\; \frac{|e_i - e_j|}{e_j} > q \\ -\frac{1}{j} & \textit{otherwise} \end{cases}$$
5) After all the project cases are examined, repeat the entire process 20 times, and remove the project cases whose average EID across the 20 trials is higher than that of the 90^{th} percentile of all the average EID calculated.

In evaluation, *FISI* showed only marginal performance improvement in its proposed study, the authors [17] believe that further analysis on dataset's characteristic will make *FISI* applicable in practical scenarios.

2) TEAK: Kocaguneli et al. [14] introduced *TEAK* (short for Test Essential Assumption Knowledge) as a method to measure if a project case should be removed from a dataset prior to building an analogy-based estimation model based on it. The removal criteria are based on whether including the project case would violate the essential hypothesis of analogy-based estimation, which is: project cases with similar charac-

teristics should consume similar amount of effort to complete their developments. Clearly, this question is analogous to the question asking whether the project cases being examined should be concerned with effort-inconsistency. Hence, project cases violating this hypothesis appear to be concerned with effort-inconsistency.

The essence of *TEAK* is a function named *Variance heuristic* [14] whose main procedure is to generate a hierarchical clustering tree over datasets. The main components of the hierarchical clustering tree are described as follows:

- Leaf nodes represent the effort values;
- Branches are formed by the level of similarity between pairing of project cases;
- The higher-level subtrees merge the lower-level subtrees using their median of similarity values within subtrees.

The type of binary tree adopted in the study of *TEAK* [14] is Greedy agglomerative clustering (*GAC*) tree [1]. *GAC* is built bottom up in a manner of greedy algorithm. Based on experimentations, the elimination process is carried out by pruning all the subtrees whose value of $\sigma^2(Effort)$ are greater than 10% of the maximum $\sigma^2(Effort)$ calculated from all the subtrees [14]. The 10% parameter value was suggested based on results of 20 randomized trials across multiple performance measures and various experimental datasets.

In [14], *TEAK* showed outstanding performance for 6 out of the 9 selected datasets. Applying *TEAK* to datasets prior to building a very simple analogy-based SEE model outperformed 8 other model-based SEE methods built using full training datasets. However, there is no evidence available that *TEAK* will also perform well with other SEE models.

C. Motivating Example

The common procedures observed between *FISI* and *TEAK* are that:

- Effort-inconsistency is observed and examined in pairs or small subsets of project cases;
- The project cases violating the effort-inconsistency are the cases in the same subset whose effort values are relatively highly different.
- Both *FISI* and *TEAK* remove all the pairs or subsets of effort-inconsistent project cases detected to make the remaining dataset more effort consistent.

Figure 1 provides an example showing that it is unnecessary to remove every single case being detected. Instead, a further analysis of each case is able to disclose which of the cases are necessary to be removed. This figure shows records of 6 raw SEE project cases of a dataset named Kemerer, commonly used in experiments to evaluate SEE models [9]. To ease the explanation of this example, we present only 6 of its project cases that were originally labelled as *Language = 1* and *Hardware = 1*, indicating that all these cases are similar by means of development language and hardware.

Applying *FISI* or *TEAK* to these project cases, the cases 9 and 11 (highlighted in Figure 1) will be removed by *FISI*

ID	Duration	KSLOC	RawFP	AdjFP	EffortMM
1	17	253.6	1217.1	1010	287.0
4	18	214.4	788.5	881	86.9
9	14	289.0	1592.9	1554	116.0
10	5	39.0	240.0	250	72.0
11	13	254.2	1611.0	1603	258.7
14	26	164.8	1347.5	1375	246.9

Fig. 1. An example of SEE datasets (a subset of the Kemerer dataset from [9] having *Language = 1* and *Hardware = 1*)

or *TEAK* due to their recorded effort values that are very different despite their similarity in characteristics. However, it is observed that, if we further examine the two cases, one by one, by assuming that the other case does not exist in the dataset, we will be able to see that only the case 9 seems to be effort-inconsistent, and we only need to remove it to achieve a higher effort-consistent dataset. That is, if we exclude case 11 from the dataset, the case 1 will become the closest neighbor of the case 9. However, the difference between the effort values of case 9 and case 1 is still very large (i.e., the 2 cases were completed with 116 and 287 man-month, respectively). Conversely, if we instead exclude the case 9 from the dataset, the closest neighbor of the case 11 will also be the case 1, but this time, the difference of the effort values of the 2 cases (258.7 and 287 man-month, respectively) is much less than that of calculated between the case 9 and 1. Based on this observation, a better approach to detect a deviant project case is to investigate if it still produces conflict with any other remaining cases after its conflicted cases detected at the first run were assumed to be nonexistent in the dataset.

III. Filter-INC: A Technique for Handling Effort-Inconsistency in SEE Datasets

To study whether the situation given in the example is a general situation in SEE, and our proposed hypothesis will generally hold true, we implement the hypothesis and name the implementation *Filter-INC* (short for a Filtering technique for data INConsistency handling). *Filter-INC* can be also viewed as an executable measure applicable to any existing effort-inconsistent filtering technique to justify whether the cases detected by them should be removed or not. The procedure of *Filter-INC* is graphically illustrated in Figure 2 and explained in general steps as follows:

1) For a project case$_i$ being investigated as a cause of effort-inconsistency using a technique T (i.e., $T_1 = FISI$):
2) *Filter-INC* initially searches for its conflict cases (denoted as a set C);
3) Then, it reruns T over the dataset but excluding all the cases in C;
4) The decision made by *Filter-INC* is that *Filter-INC* will leave the case$_i$ in the dataset if it is no longer concerned by T; otherwise, the case$_i$ appears to be the cause of effort-inconsistency with a need of removal.

The rest of this section explains the applications of *Filter-INC* on *FISI* and *TEAK*, respectively.

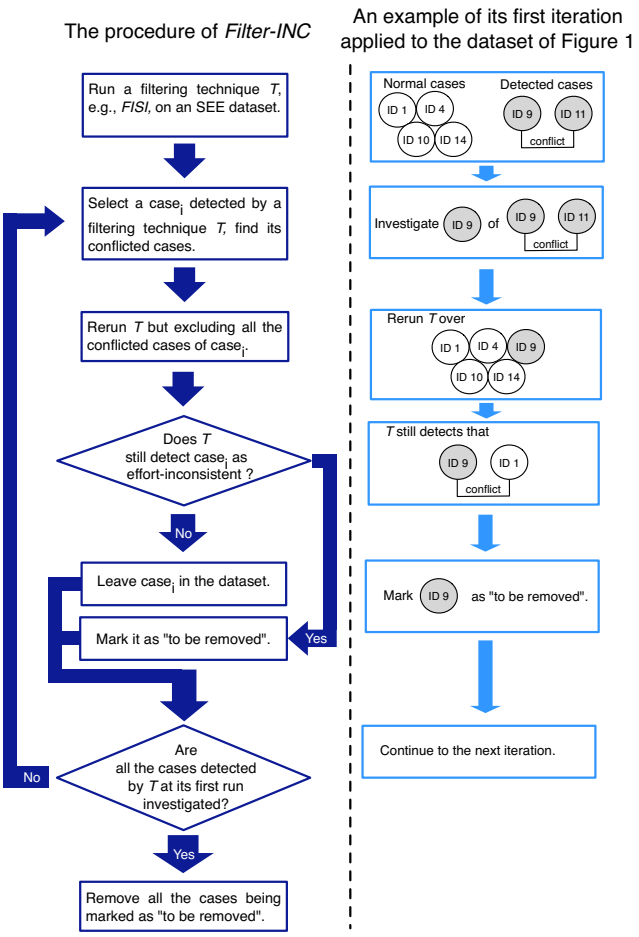

Fig. 2. Graphical explanation with example of the procedure of *Filter-INC*

A. Application of Filter-INC in FISI

Since *FISI* assesses similarity between project cases using k-nearest neighbor, where k is set to 3; conflict cases can be simply identified using this k-nearest neighbor scheme. Let a case$_i$ denote a project case being concerned by *FISI* as effort-inconsistent in its first run, i.e., EID$_i$ is falling in the region of 90^{th} percentile of the whole EID. Considering a case$_i$ and a case$_j$ are within the 3-nearest neighbors of each other and have been classified as effort-inconsistent data by *FISI*, we make the assumption that if one of the case (case$_j$) is removed from the dataset and case$_i$ is still detected by *FISI* as effort-inconsistent after a second run of test, then surely, case$_i$ is effort-inconsistent. In this way, *Filer-INC* will remove case$_i$ from the dataset due to being the cause of effort-inconsistency; otherwise, it will suggest to leave the case$_i$ in the dataset. In steps, applying *Filter-INC* to FISI is proceeded as given below:

1) Select a case$_i$ from the set of project cases detected by *FISI* as effort-inconsistent cases;
2) Let C denote a set consisting of up to 3 project cases, where each case in C and the case$_i$ are mutually 3-nearest neighbors of each other;
3) Rerun *FISI* over the dataset but excluding C;
4) If this time the EID$_i$ is no longer falling in the 90^{th} percentile of the whole EID, leave the case$_i$ in the dataset; otherwise, mark it as "to be removed";
5) Repeat the above steps for all the cases falling in the 90^{th} percentile region at the first run of *FISI*.
6) After all the cases are examined, remove all the marked cases from the datasets.

B. Application of Filter-INC in TEAK

Let $GAC_{height=1}$ denote a subtree whose height is equal to 1. Following the procedure of *TEAK* [14], the subtrees to be concerned with inconsistency are those whose overall $\sigma^2(Effort)$ are greater than 10% of the maximum $\sigma^2(Effort)$ calculated from all the $GAC_{height=1}$. Applying *Filter-INC* to *TEAK*, all the project cases in these subtrees will further be analyzed by *Filter-INC*.

The original *TEAK* [14] removes all these project cases in all the detected subtrees. On the contrary, based on our hypothesis, *Filter-INC* questions which of the cases in each subtree are probably deviants and needs to be removed to make the dataset more effort-consistent. In steps, applying *Filter-INC* to *TEAK* is proceeded as follows:

1) Select a case$_i$ from the set of projects detected by *TEAK* as effort-inconsistent cases;
2) Let C denote a set containing project cases in the same $GAC_{height=1}$ as the case$_i$;
3) Rebuild *TEAK* over the dataset but excluding C;
4) If this time the case$_i$ is no longer detected by *TEAK*, leave it in the dataset; otherwise, mark it as "to be removed";
5) Repeat the above steps for all the cases being detected by *TEAK*.
6) After all the cases are examined, remove all the marked cases from the datasets.

IV. Evaluation Methodology

A. Datasets

The evaluation is carried out using 8 industrial software project datasets, selected from the list of standard benchmark dataset commonly used in experiments to evaluate SEE models [14], [16], [10], [13]. All the selected datasets are made available in the tera-Promise software repository [18] to allow complete replication of SEE studies. Table I provides summary of the 8 datasets.

Kemerer is a small dataset consisting of 15 project cases from large business [9]. Cocomo81-e and Cocomo81-so [2] are subsets of the well-known Cocomo81 dataset, which is a collection of software projects in NASA. Cocomo81-e concerns only project cases developed in embedded mode, and Cocomo81-so concerns project cases developed in semi-detached or organic modes, respectively. Decharnais-Cobols and Decharnais-4GL are subsets of the most well-known dataset in SEE studies, the Desharnais dataset. This Decharnais

TABLE I
THE 8 INDUSTRIAL DATASETS CONSISTING OF 244 SOFTWARE PROJECT CASES (F: # FEATURES, N: # PROJECT CASES, I: % OF EFFORT-INCONSISTENT CASES DETECTED BY *TEAK*, AND E: EFFORT)

	Dataset	F	N	I	E (h: hours, m: months)		
					Unit	Med	Max
1	Kemerer	8	15	73.33	m	130	1107
2	Cocomo81-e	18	29	10.71	m	354	11400
3	Cocomo81-so	18	35	20.00	m	50	6400
4	Desharnais-Cobols	10	67	25.37	h	3913	23940
5	Desharnais-4GL	10	10	40.00	h	1123	5880
6	Nasa93-c1	23	12	41.67	m	66	360
7	Nasa93-c2	23	37	13.51	m	82	1350
8	Nasa93-c5	23	39	15.39	m	571	8211

dataset is a collection of software projects developed in Canada and collected by using function points approach [22]. Decharnais-Cobols concerns project cases developed in Basic Cobol or Advanced Cobol languages, and Decharnais-4GL concerns project cases developed in 4GL languages. Finally, the Nasa93-c1, Nasa93-c2, and Nasa93-c5 datasets are subsets split from the Nasa93 dataset [2], which is another collection of NASA software projects. For these 3 datasets, subsetting is taken by the centers where project cases were developed.

Applying data subsetting to the Cocomo81, Desharnais, and Nasa93 datasets is a common practice in SEE studies [14], [16], [10], [13]. It is mainly based on the assumption of locality, such that characteristics of different subdivisions in the same company are naturally diversified. Of the 7 sub datasets, we merged the Cocomo81 project cases in semi-detached and organic modes into one single dataset because of their overlapped productivity distribution (tested by applying Kernel density plots [11] to the $Effort/Size$ variables of the two datasets). The same reason is also applied for the Deshanais-Cobols dataset.

The third column of Table I, indicated by I, illustrates the percentage of the effort-inconsistent cases detected by *TEAK* for each dataset. The percentage of cases detected by *FISI* is not presented here because *FISI* always detects 10% of the total cases as being effort-inconsistent. Observing these numbers, the percentage of the effort-inconsistent cases of the 8 datasets are in the range between 10.71% and 73.33% (the median is 22.68%). This indicates that effort-inconsistent cases are not rare in practice, and therefore, an effective approach for their handling can be practically useful.

B. Sampling Approach

For each dataset, sampling is applied to split training/test instances. The sampling approach of our choice is leave-one-out [6]. It is selected following the conclusion drawn in a study by Kocaguneli and Menzies [13], stating that the leave-one-out approach should be used in empirical software engineering studies mainly because the results generated based on it are deterministic and reproducible. The leave-one-out approach splits $n-1$ project cases from a dataset of n project cases, and applying an SEE model to the $n-1$ case to estimate an effort for the 1 other case.

C. SEE Models

SEE models selected in our experiment are commonly used in SEE literature [10], [15]. Precisely, we selected 3 SEE models as follows: classification and regression trees (*CART*), 1-nearest neighbor analogy-based model (*ABE0-1NN*), and 5-nearest neighbor analogy-based model (*ABE0-5NN*). These 3 models are of our choice mainly because they were the 3 highest performers in the study where conclusion stability of SEE models was successfully concluded [10]. Due to the limitation of space, and since all of these models are very common in SEE studies, please refer to the appendix of any of the following SEE studies for their explanation [10], [15].

D. Treatments

Treatments of our experiments consist of 5 choices of model adjustments: 2 choices between *FISI* and *TEAK* $\times$ 2 choices of having or not having *Filter-INC* applied $+$ 1 choice of applying no technique at all. Table II depicts the terms we call each treatment in the rest of this paper.

TABLE II
TERMS DENOTE TREATMENTS IN OUR EXPERIMENTS

	Term	Filtering technique	Apply Filter-INC?
1	FISI	FISI	no
2	Filter-INC(FISI)	FISI	yes
3	TEAK	TEAK	no
4	Filter-INC(TEAK)	TEAK	yes
5	NONE	-	no

Applying all these treatments to 3 SEE models will generate in total of 15 variants of SEE models. Let a term "an estimator" denote a pair of one SEE model and one treatment (e.g., *CART* and *FISI*).

E. Performance Measures

Applying an estimator to a dataset with n project cases will generate n estimated effort values. Let e_i and e'_i denote an actual and an estimated effort value for case$_i$, respectively. The performance measures we used to assess and compare each estimator are taken from the list of robust performance measures recommended by Foss et al. [7] in their assessment of multiple measures commonly used in SEE studies. Precisely, 7 measures are chosen in our study. A detailed explanation of the 7 performance measures are as follows:

The absolute residual (*AR*) is the absolute difference between e_i and e'_i. The overall error of *AR* is commonly derived as the Mean Absolute Error (*MAR*), and Median Absolute Error (*MdAR*):

$$MAR = mean(\forall i\ |e_i - e'_i|), \quad MdAR \quad = median(\forall i\ |e_i - e'_i|)$$

Derived from *AR*, relative error measures such as *MMRE* and *Pred(25)* [7] are frequently used in the literature on SEE.

However, based on a detailed analysis undertaken over these measures, Foss et al. [7] rather suggested that most of them are biased and unreliable. According to Foss et al., of the large set of these measures, 2 of them are considered sufficiently trustworthy. The two measures are the Mean balanced relative error (*MBRE*) and the Mean inverted balanced relative error (*MIBRE*):

$$MBRE = mean\left(\forall i \ \frac{|e_i - e'_i|}{min(e_i, e'_i)}\right),$$
$$MIBRE = mean\left(\forall i \ \frac{|e_i - e'_i|}{max(e_i, e'_i)}\right)$$

The other 3 error measures are variants of standard deviation: Standard Deviation *(SD)*, Relative Standard Deviation *(RSD)*, and Logarithmic Standard Deviation *(LSD)*, their formulas are depicted as:

$$SD = \sqrt{\frac{\sum_{i=1}^{n}(e_i - e'_i)^2}{n-1}}, \qquad RSD = \sqrt{\frac{\sum_{i=1}^{n}\left(\frac{e_i - e'_i}{ss_i}\right)^2}{n-1}},$$
$$LSD = \sqrt{\frac{\sum_{i=1}^{n}\left(E_i - \left(-\frac{S^2}{2}\right)\right)^2}{n-1}}$$

In *RSD* and *LSD*, ss_i is a single software size variable (e.g., adjusted Function points) of a case$_i$, E_i is given by $ln(e_i) - ln(e'_i)$, and S is the sample variance of E, given by $\frac{1}{n-1}\sum_{i=1}^{n}(E_i - \bar{E})^2$.

To ensure the robustness of the results guided by multiple error measures, we applied a redundancy analysis to the estimation results across different measures and selected only those being uncorrelated to any others. The analysis is provided by *redun* function in the *rms* R package [8]. In particular, MAR, MBRE, and SD were suggested to be removed as they can be represented by MdAR, MIBRE, and LSD, respectively. Hence, only 4 remaining error measures: MdAR, MIBRE, RSD, and LSD are used throughout the study.

F. Performance Comparison

One of the most common evaluation procedures used in SEE studies is *win-tie-loss statistics*. Adopting the statistics in our experiment, a pairwise-based evaluation is applied to all the possible pairs of estimators. The overall performance of an estimator is assessed by 3 counters, *wins*, *ties*, *losses*. Comparing between 2 estimators, if the error distributions of the 2 estimators are not significantly different, as confirmed by a statistical test method named Brunner's test [5], the *ties* counters of the 2 estimators will be increased by 1; Otherwise, the *wins* counter of the estimator with higher performance will increase, and the *losses* counter of the other estimator will increase.

V. Results

Our first experiment evaluates the effects of applying *Filter-INC* on the estimation performance of *FISI* and *TEAK*. The count based on the *win-tie-loss* statistics was adopted in this experiment by comparing between *NONE*, a technique T, and *Filter-INC(T)* for every single dataset and SEE model. For example, to observe the performance change as a result of applying *Filter-INC* to *FISI*, there are 8 comparisons involved for each estimator: 4 error measures $\times$ 2 other treatments. Table III illustrates the results by means of total *losses* generated across all the selected datasets and SEE models. *Losses* is selected in this main experiment following the setup in 2 prominent studies where 2 long debates in SEE were successfully concluded [10], [15].

TABLE III
PERFORMANCE BY MEANS OF *losses*. LOWER *losses* VALUE INDICATES HIGHER PERFORMANCE. THE ENTRIES BEING HIGHLIGHTED IN LIGHT GRAY INDICATE THAT APPLYING *Filter-INC* RESULTED IN A **MORE** DESIRABLE PERFORMANCE, AND THOSE BEING HIGHLIGHTED IN DARK GRAY INDICATE THE OTHERWISE.

Dataset	Model	Total losses			
		FISI	Filter-INC (FISI)	TEAK	Filter-INC (TEAK)
Kemerer	CART	0	0	0	0
	ABE0-1NN	0	0	4	1
	ABE0-5NN	0	0	1	0
Cocomo81 -e	CART	2	0	2	0
	ABE0-1NN	1	0	2	0
	ABE0-5NN	2	1	2	0
Cocomo81 -so	CART	8	0	2	0
	ABE0-1NN	0	0	2	0
	ABE0-5NN	0	0	2	0
Desharnais -Cobols	CART	0	0	2	0
	ABE0-1NN	0	0	2	0
	ABE0-5NN	0	0	1	0
Desharnais -4GL	CART	0	0	0	0
	ABE0-1NN	0	0	0	0
	ABE0-5NN	3	3	0	0
Nasa93-c1	CART	0	0	8	0
	ABE0-1NN	0	0	6	0
	ABE0-5NN	0	0	8	0
Nasa93-c2	CART	0	0	1	1
	ABE0-1NN	0	0	2	0
	ABE0-5NN	0	0	1	0
Nasa93-c5	CART	0	0	0	0
	ABE0-1NN	1	0	0	2
	ABE0-5NN	0	0	0	0

Table III illustrates the results by means of total *losses* generated across all the selected datasets and SEE models. For any entry in this table, the minimum *losses* is 0 (indicating that the treatment was never outperformed by *NONE* and the other treatment being compared), and the maximum *losses* is 8 (indicating that it was significantly outperformed in all the comparisons). In short, lower *losses* value indicates higher performance. From this table, 3 findings are outstandingly noteworthy:

- Only one single comparison shows that applying *Filter-INC* to *FISI* or *TEAK* resulted in a reduced performance.
- The most outstanding results are shown in the Nasa93-c1 dataset where applying *Filter-INC* to *TEAK* significantly improved the performance from being all (8) *losses* to no single (0) *losses*.

TABLE IV
PERFORMANCE IN TERMS OF *wins*, *losses*, AND *wins-losses* AGGREGATED FROM ALL THE 8 DATASETS. HIGHER *wins* AND *wins-losses* INDICATE HIGHER PERFORMANCE, WHERE AS LOWER *losses* INDICATE HIGHER PERFORMANCE. THE RANKS ARE ORDERED BY *losses*

	Model	Treatment	wins	losses	wins-losses
1	ABE0-5NN	Filter-INC(TEAK)	66	10	56
2	ABE0-5NN	FULL	58	20	38
3	CART	Filter-INC(TEAK)	41	20	21
4	ABE0-1NN	Filter-INC(FISI)	66	28	38
5	ABE0-5NN	Filter-INC(FISI)	48	36	12
6	CART	FULL	41	37	4
7	CART	Filter-INC(FISI)	48	37	11
8	ABE0-1NN	FULL	57	40	17
9	ABE0-1NN	Filter-INC(TEAK)	57	43	14
10	ABE0-5NN	TEAK	30	43	-13
11	ABE0-1NN	FISI	55	44	11
12	ABE0-5NN	FISI	47	46	1
13	CART	FISI	14	72	-58
14	ABE0-1NN	TEAK	44	108	-64
15	CART	TEAK	15	109	-94

- The total *losses* are generally reduced to only 0 or 1 after *Filter-INC* has been applied. This indicates the optimum or the very close to optimum performance.

Out of the 24 comparisons across 8 datasets and 3 SEE models, applying *Filter-INC* to *FISI* never resulted in a fallback performance; however, only 5 comparisons showed improved performance. On the other hand, applying *Filter-INC* to *TEAK*, 16 comparisons indicated improvement, whereas there was a comparison that resulted in a fallback performance. This can be inferred that applying *Filter-INC* to *TEAK* seems more likely to improve the performance than applying it to *FISI*. Furthermore, out of the 5 comparisons where *FISI* and *TEAK* originally attained 50% *losses* or more (i.e., equal to or higher than 4 *losses*), the losses of these comparisons were reduced to only 0 or 1 after *Filter-INC* was applied. This indicates that application of *Filter-INC* has a clear positive impact on the estimation performance.

Subsequently, we carried out our second experiment to determine and recommend of which estimators are generally the high performers. In this experiment, each single estimator was compared to 14 other estimators (3 models $\times$ 5 treatments $-$ 1) subject to 4 error measures and 8 datasets. Hence the maximum *wins* or *losses* are both equal to 14 $\times$ 4 $\times$ 8 = 448. Table IV provides the results in terms of the total *wins*, *losses*, and *wins-losses*. Ranks in this table are ordered by *losses*. Examining each model, the results indicate that *Filter-INC(FISI)* consistently outperformed *FISI*, and *Filter-INC(TEAK)* consistently outperformed *TEAK*. Of the 3 models, pairing *Filter-INC(TEAK)* with *ABE0-5NN* or *CART* outperformed all the other 4 treatments, where *ABE0-1NN* showed its best performance when being paired with *Filter-INC(FISI)*. Overall, adopting *Filter-INC(TEAK)* with *ABE0-5NN* yielded the best performance in all terms of *wins*, *losses* and *wins-losses*. The results of Table IV are in agreement with that of Table III, showing an encouraging possibility to transfer *Filter-INC* to the real-world application of SEE for handling the problem of effort-inconsistency in SEE datasets.

VI. DISCUSSIONS

A. Generalization

Generalizing from the results with statistical significance confirmed, the success of *Filter-INC* confirms the association between estimation performance and data quality in terms of effort-inconsistency. After the proposal of *Filter-INC*, we suggest revisiting an analysis of the influential factors of the effort. After the removal of effort inconsistent cases, changes in correlations between project features and the effort, or of pairs between project features themselves, may allow us to capture more numbers and obtain more variations of the influential factors, leading to a potential reduction in estimation error after these factors are exploited.

From the hypothesis of *Filter-INC*, not only data inconsistency has been proved to be an important data quality attribute affecting accuracy of SEE but data diversity has been shown to be important as well. Considering the difference between *Filter-INC* and conventional methods, such as *FISI* and *TEAK*, the essential difference is the decision to leave some project cases in the datasets, clearly indicating datasets preprocessed by *Filter-INC* should have higher diversity. Hence, a further study with more focus on data diversity is therefore suggested to examine whether the association between the estimation performance and the degree of data diversity can also be concluded. This data quality attribute is what appealed to us because, referring to a survey study by Bosu and MacDonnell [3], very few studies on empirical software engineering have addressed the issue, whereas our finding in this study shows a high likelihood that it should be associated with accuracy of learning models commonly adopted in the literature on empirical software engineering.

B. Importance of a technique like Filter-INC

Different from several other data quality problems, such as missing values and redundancy, where a more transparent and deliberate data collection strategy is the common suggestion for their handling [4], effort-inconsistency does not seem to benefit from it. This is because project data that are correctly collected may coincidently have been completed with diversified amount of effort. In this way, only filtering techniques seem to be applicable for their handling.

Due to the problem of space limitation, we are unable to show statistics of the effort-inconsistent records in our experimental datasets. Nevertheless, the results indicate a successful application of *Filter-INC* in many datasets. It may be implied that these records are commonly contaminated in the real-world SEE datasets, and a successful attempt to correctly remove them will clearly contribute to the increase in estimation performance.

C. Threats to Validity

Internal validity: We believe that the parameters of our choice configured in our experiments are of high reliability. For example, the choice of adopting the leave-one-out sampling approach does enable a fully replication of the entire experimental method, where its alternative popular choice namely n-way cross-validation does not [13].

External validity: Number of total project cases we used in our experiments are slightly higher than that of commonly used in SEE studies, as reported by Kitchenham in [12]. Precisely, our experiments used the total of 244 project cases where the report by Kitchenham showed that the median value of project cases in SEE studies is 186.

Construct validity: We carefully selected the error measures and the statistical significant test method from the list of robust measures and method as recommended by Foss et al. [7] and Kitchenham et al. [11], respectively. And to ensure robustness and stability of the results, we applied a redundancy analysis to the list of error measures and selected only 4 of which that are statistically independent to be used throughout the study.

VII. Conclusion

This study proposes a novel technique known as *Filter-INC* for a better handling of the effort-inconsistency problem [17] in software effort estimation (SEE) datasets. The hypothesis supporting *Filter-INC* suggests a detection and a removal of only the deviant project cases from datasets. We define the deviant project cases as the cases that have been detected as effort-inconsistent in the first detection, and it is still detected as effort-inconsistent again after a temporary removal of its conflicted cases in the second detection. Empirically validated, *Filter-INC* provided a significantly improvement in the estimation accuracy. We have compared *Filter-INC* with 2 existing techniques (*FISI* [17] and *TEAK* [14]), whose hypotheses behind their procedures do not contain the notion of deviant cases. The evaluation results of *Filter-INC* allow us to conclude the following findings:

- Utilizing the key hypothesis given in *Filter-INC* provides a very effective approach to handle the problem of effort-inconsistency in SEE datasets.
- Applying *Filter-INC* to existing techniques, such as *TEAK* [14], shows a significantly improved performance compared in our experiments.
- Besides the problem related to data inconsistency, data diversity is very encouraging to be further studied as it has a very high likelihood of being associated with estimation performance as well;

The success of *Filter-INC* indicated by our evaluation results discloses a new and interesting research direction to tackle data quality problems in SEE. Utilizing its hypothesis alongside existing techniques, the essence of Filter-INC that analyzes inter-relationship between project cases will help us to establish a more effective approach to improve the quality of SEE datasets, ultimately leading to a better estimation accuracy of SEE models.

Acknowledgements

This research was supported by JSPS KAKENHI Grant number 26330086, was conducted as a part of the Program for Advancing Strategic International Networks to Accelerate the Circulation of Talented Researchers: Interdisciplinary Global Networks for Accelerating Theory and Practice in Software Ecosystem, and was supported in part by the City University of Hong Kong research fund (Project number 7200354, 7004222, 7004474, 7004683, and 9042328).

References

[1] D. Beeferman and A. Berger, "Agglomerative clustering of a search engine query log," in *Proc. Sixth ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, 2000, pp. 407–416.

[2] B. W. Boehm, *Software Engineering Economics*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1981.

[3] M. F. Bosu and S. G. MacDonell, "Data quality in empirical software engineering: A targeted review," in *Proc. 17th Int'l Conf. Evaluation and Assessment in Software Eng.*, 2013, pp. 171–176.

[4] ——, "A taxonomy of data quality challenges in empirical software engineering," in *Proc. 22nd Australasian Software Eng. Conf.*, 2013, pp. 97–106.

[5] E. Brunner, U. Munzel, and M. L. Puri, "The multivariate nonparametric behrens–fisher problem," *J Statistical Planning and Inference*, vol. 108, no. 1, pp. 37–53, 2002.

[6] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Machine Learning Research*, vol. 7, pp. 1–30, 2006.

[7] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit, "A simulation study of the model evaluation criterion mmre," *IEEE Trans. Software Eng.*, vol. 29, no. 11, pp. 985–995, 2003.

[8] F. E. Harrell Jr, M. F. E. Harrell Jr, and D. Hmisc, "Package rms," *Vanderbilt University*, p. 229, 2016.

[9] C. F. Kemerer, "An empirical validation of software cost estimation models," *Commun ACM*, vol. 30, no. 5, pp. 416–429, 1987.

[10] J. Keung, E. Kocaguneli, and T. Menzies, "Finding conclusion stability for selecting the best effort predictor in software effort estimation," *Automated Software Eng.*, vol. 20, no. 4, pp. 543–567, 2013.

[11] B. Kitchenham, L. Madeyski, P. Brereton, S. Charters, S. Gibbs, and A. Pohthong, "Robust statistical methods for empirical software engineering," *Empirical Software Eng.*, pp. 1–52, 2016.

[12] B. Kitchenham, E. Mendes, G. H. Travassos *et al.*, "Cross versus within-company cost estimation studies: A systematic review," *IEEE Trans. Software Eng.*, vol. 33, no. 5, pp. 316–329, 2007.

[13] E. Kocaguneli and T. Menzies, "Software effort models should be assessed via leave-one-out validation," *J. Systems and Software*, vol. 86, no. 7, pp. 1879–1890, 2013.

[14] E. Kocaguneli, T. Menzies, A. Bener, and J. W. Keung, "Exploiting the essential assumptions of analogy-based effort estimation," *IEEE Trans. Software Eng.*, vol. 38, no. 2, pp. 425–438, 2012.

[15] E. Kocaguneli, T. Menzies, and J. Keung, "On the value of ensemble effort estimation," *IEEE Trans. Software Eng.*, vol. 38, no. 6, pp. 1403–1416, 2012.

[16] E. Kocaguneli, T. Menzies, and J. W. Keung, "Kernel methods for software effort estimation," *Empirical Software Eng.*, vol. 18, no. 1, pp. 1–24, 2013.

[17] T. K. Le-Do, K.-A. Yoon, Y.-S. Seo, and D.-H. Bae, "Filtering of inconsistent software project data for analogy-based effort estimation," in *Proc. 34th Annual Computer Software and Applications Conf.*, 2010, pp. 503–508.

[18] T. Menzies, M. Rees-Jones, R. Krishna, and C. Pape, "The promise repository of empirical software engineering data," July 2015.

[19] P. Phannachitta, A. Monden, J. Keung, and K. Matsumoto, "Case consistency: a necessary data quality property for software engineering data sets," in *Proc. 19th Int'l Conf. Evaluation and Assessment in Software Eng.*, 2015, p. 19.

[20] Y.-S. Seo and D.-H. Bae, "On the value of outlier elimination on software effort estimation research," *Empirical Software Eng.*, vol. 18, no. 4, pp. 659–698, 2013.

[21] M. Shepperd, "Software project economics: A roadmap," in *Proc. Future of Software Eng.*, 2007, pp. 304–315.

[22] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," *IEEE Trans. Software Eng.*, vol. 23, no. 11, pp. 736–743, 1997.