DATA CLEANING TECHNIQUES FOR SOFTWARE ENGINEERING DATA SETS

A thesis submitted towards the degree of

Doctor of Philosophy

by

GERNOT ARMIN LIEBCHEN

School of Information Systems, Computing and Mathematics

Brunel University

October 2010

Abstract

Data quality is an important issue which has been addressed and recognised in research communities such as data warehousing, data mining and information systems. It has been agreed that poor data quality will impact the quality of results of analyses and that it will therefore impact on decisions made on the basis of these results. Empirical software engineering has neglected the issue of data quality to some extent. This fact poses the question of how researchers in empirical software engineering can trust their results without addressing the quality of the analysed data. One widely accepted definition for data quality describes it as 'fitness for purpose', and the issue of poor data quality can be addressed by either introducing preventative measures or by applying means to cope with data quality issues. The research presented in this thesis addresses the latter with the special focus on noise handling.

Three noise handling techniques, which utilise decision trees, are proposed for application to software engineering data sets. Each technique represents a noise handling approach: robust filtering, where training and test sets are the same; predictive filtering, where training and test sets are different; and filtering and polish, where noisy instances are corrected. The techniques were first evaluated in two different investigations by applying them to a large real world software engineering data set. In the first investigation the techniques' ability to improve predictive accuracy in differing noise levels was tested. All three techniques improved predictive accuracy in comparison to the do-nothing approach. The filtering and polish was the most successful technique in improving predictive accuracy. The second investigation utilising the large real world software engineering data set tested the techniques' ability to identify instances with implausible values. These instances were flagged for the purpose of evaluation before applying the three techniques. Robust filtering and predictive filtering decreased the number of instances with implausible values, but substantially decreased the size of the data set too. The filtering and polish technique actually increased the number of implausible values, but it did not reduce the size of the data set.

Since the data set contained historical software project data, it was not possible to know the real extent of noise detected. This led to the production of simulated software engineering data sets, which were modelled on the real data set used in the previous evaluations to ensure domain specific characteristics. These simulated versions of the data set were then injected with noise, such that the real extent of the noise was known. After the noise injection the three noise handling techniques were applied to allow evaluation. This procedure of simulating software engineering data sets combined the incorporation of domain specific characteristics of the real world with the control over the simulated data. This is seen as a special strength of this evaluation approach.

The results of the evaluation of the simulation showed that none of the techniques performed well. Robust filtering and filtering and polish performed very poorly, and based on the results of this evaluation they would not be recommended for the task of noise reduction. The predictive filtering technique was the best performing technique in this evaluation, but it did not perform significantly well either.

An exhaustive systematic literature review has been carried out investigating to what extent the empirical software engineering community has considered data quality. The findings showed that the issue of data quality has been largely neglected by the empirical software engineering community.

The work in this thesis highlights an important gap in empirical software engineering. It provided clarification and distinctions of the terms noise and outliers. Noise and outliers are overlapping, but they are fundamentally different. Since noise and outliers are often treated the same in noise handling techniques, a clarification of the two terms was necessary.

To investigate the capabilities of noise handling techniques a single investigation was deemed as insufficient. The reasons for this are that the distinction between noise and outliers is not trivial, and that the investigated noise cleaning techniques are derived from traditional noise handling techniques where noise and outliers are combined. Therefore three investigations were undertaken to assess the effectiveness of the three presented noise handling techniques. Each investigation should be seen as a part of a multi-pronged approach.

This thesis also highlights possible shortcomings of current automated noise handling techniques. The poor performance of the three techniques led to the conclusion that noise

handling should be integrated into a data cleaning process where the input of domain knowledge and the replicability of the data cleaning process are ensured.

Publications Resulting from the Work Presented in this Thesis

The work presented in this thesis resulted in the publication of four conference papers [108, 110, 111, 109].

Contents

List of Figures

List of Tables

Acknowledgments

First and foremost I would like to thank my supervisor Martin Shepperd, not only for his academic guidance and assistance, but also for his patience and personal support. I am truly grateful.

I would also like to thank Bhekisipho Twala. His input and supervision was crucial for large parts of this thesis. Ngiyabonga!

Thanks to Michelle Cartwright and Steve Counsell for their work as second supervisors.

Many thanks to Traci Hall who showed a lot of patience and goodwill. I hope I was not too testing.

Thanks too to Mark Stephens and Ambikesh Jayal for their practical assistance in the investigations carried out for this thesis, and thanks to Rahul Premraj for his help in the early stages of this PhD.

I would also like to thank my grandfather, Gerhard Liebchen for his interest and financial support.

Chapter 1

Introduction

“[...]they can be analyzed to unearth costly corporate habits; they can be manipulated to divine future trends. Just one problem: Those huge databases may be full of junk. . . .”[193]

1.1 Importance of Data Quality

In empirical software engineering the most important input is data. Data are used to predict, discover and to decide on new strategies. They are also used to indicate that new strategies are working, or what impact new techniques have. It is interesting to see then, that data quality in empirical software engineering appears to be somewhat neglected in publications and even in data analyses. It is all the more astonishing since, as De Vaux and Hand [37] stated, 60-95% of the effort of data analysis is spent on the cleaning of data. This poses the question: *Is data quality not as important in empirical software engineering?*

In other research areas like information systems and data mining the impact of poor data has been recognised as an issue which needs to be addressed by database designers and data users alike. Redman [147] for instance stated that poor data quality is an issue which impacts on "all segments of the economy: companies, governments, and academia and their customers", and Wand and Wang [193] warned of the "severe impact of poor data quality on the effectiveness of an organisation". Therefore poor data quality is likely to have a 'severe impact' on empirical software engineering too!

Redman stated that low data quality increases operational costs and can impact on operations, tactics and strategies. As operational impacts he listed lowered customer satisfaction, increased cost and lowered employee satisfaction. Tactical impacts affect decision making, implementation and re-engineering. They also increase organisational mistrust. Strategic impacts result in difficulties in setting and executing strategies. They contribute to issues of ownership, affect the ability to align organisations with strategies, and they divert management attention. What does that mean for empirical software engineering? The customers of empirical software engineers are practitioners. If the results researchers are producing is of low quality, practitioners will ignore the research community since it does not provide meaningful answers, rendering the researchers' efforts pointless exercises.

1.2 What is Data Quality?

The most widely used definition of data quality defines it as “fitness for purpose” [194, 147, 175, 63, 145]. This definition for data quality is derived from a more general quality definition as used by Crosby for instance [34]. Since this purpose is subjective and important to consider, data quality’s characteristics or dimensions are subjective too [147, 194], and cannot be assessed independent of the people who use the data [175]. This means that the domain the data are used in has to be an important consideration. There does not appear to be a general agreement on the dimensions of data quality [195]. Whilst Wand and Wang [193] define accuracy, completeness, consistency, and timeliness as dimensions of data quality; Redman [147] lists more than 10 dimensions. Redman [147] made the trenchant observation that his list cannot be comprehensive since as indicated above data quality dimension depend on the user’s view of the data, pointing towards a reason for this lack of consensus about data quality dimensions. Redman categorised his dimensions into four groups:

(i) Dimensions related to the data model,

(ii) Dimensions related to the data values,

(iii) Dimensions related to data presentation and

(iv) Dimensions related to information technology.

Redman also mentioned a possible fifth group which includes dimensions related to the enterprise level of an organisation. From these groups Redman categorised issues of data quality:

- “Issues associated with data views (the models of the real world captured in the data), such as relevancy, granularity, and level of detail.
- Issues associated with data values, such as accuracy, consistency, currency, and completeness.
- Issues associated with the presentation of data, such as the appropriateness of the format, ease of interpretation, and so forth.

- Other issues such as privacy, security, and ownership." [148]

In empirical software engineering the analysis of data is often based on historical data provided by industry [66]. Since most analysts do not have influence on the 'data views', the presentation of the data and issues of privacy, security, and ownership, and since these issues do not generally pose problems to researchers due to their academic background and education, the issues associated with the data values are of special interest to empirical software engineering research.

Timeliness and currency, that is, how valid historical data is at a given point in time, are interesting issues for empirical software engineering, considering that old project data might no longer be relevant any more, and predictions or decisions based on this data could be compromised by this non-currency or untimeliness. For example, research suggests that productivity changes over time [144]. Therefore historical software productivity data might not reflect current software processes leading to inaccurate conclusions.

De Vaux and Hand [37] also pointed to timeliness when they mention that data quality can deteriorate due to transformations, changed definitions and changed understanding. In contrast Orr stated, "if data quality is a function of its use, there is only one sure way to improve data quality-improve its use!" According to this statement the usage of data promotes its quality revision.

In terms of the quality of a single data item De Vaux and Hand [37] simply categorised data quality issues, or "bad data" (as De Vaux and Hand name them), as missing or distorted data. Missing data can easily be identified, but distortions or noise are less easy to discover. Whilst it might be possible to demonstrate the presence of noise, the absence of noise cannot be proven. Noise can be introduced and is created in all phases of data evolution; during data collection, during preliminary analyses and in the data modelling phase. Noise can be created by distorting data from the outset during data collection. It may also be introduced whilst transcribing, transferring, merging or copying the data, or, as mentioned above, data quality issues might be introduced due to the deterioration of the data. This means that analysts should at least be suspicious of the data they are using.

1.3 What is Noise?

Whilst there is consensus about the data quality definition of “fitness for purpose”, there is no consensus about the definition of noise. Manago and Kodratoff [118] stated that “noise is present when a knowledge base does not truly reflect the environment we want to learn from”. They are indicating that the causes of noise lead analysts to build inaccurate models. According to their definition, noise is “wrong information, lack of information or unreliable information”. The term “unreliable information” is interesting, since the information is not incorrect, but “unreliable”. Manago’s and Kodratoff’s definition is also interesting since it incorporates missingness. Thus their definition of noise includes any instance which could pose problems for a machine learner.

Brodley and Friedl [23, 24] declared that noise can be identified because it goes “against the ’laws of the domain’”. This is slightly confusing since it overlaps with their definition of outliers. They defined an outlier as an instance “that does not follow the same model as the rest of the data, appearing as though it comes from a different probability distribution”. This definition of noise is also adopted in a later paper by Brodley and Friedl [25]. Gamberger et al. [58] distinguished between two types of errors, systematic and random, where random errors are seen as noise. In [57] Gamberger and Lavrač also defined only random errors as noise, but in [59] Gamberger et al. used the term noise in a broader sense and included not only random errors but also outliers, which is similar to Brodley’s and Friedl’s defintion of noise. The conflation of the terms noise and outliers is typical in the machine learning domain, since both issues pose problems to machine learners. Brodley and Friedl attempt to solve this problem by searching for instances which are “outliers in *any* model” [23, 24, 25]. For this purpose Brodley and Friedl used several outlier detection algorithms and combined their results according to majority and consensus rules, where for the consensus rule all algorithms have to identify an instance as outlier in order to be considered as noise, and where for the majority rule the majority of algorithms have to identify an instance as outlier in order to be considered as noise. [1]

[1] Brodley and Friedl’s approach will be discussed again in Chapter 2.

The issue arising, considering these views on noise and outliers, is whether outliers are poor data quality instances. Are they 'bad data' as defined by De Vaux and Hand [37]? They are not missing, and if data are exceptional instances as accepted by Brodley and Friedl they are not necessarily inaccurate. Noise can be seen as "unwanted disturbance" of signals or data [124, 70]. Outliers do not have to be 'unwanted'. If they are 'true' instances they can contain valuable information about new trends in a domain. Therefore the distinction between 'true' instance and inaccurate instance is important, depending on the domain.

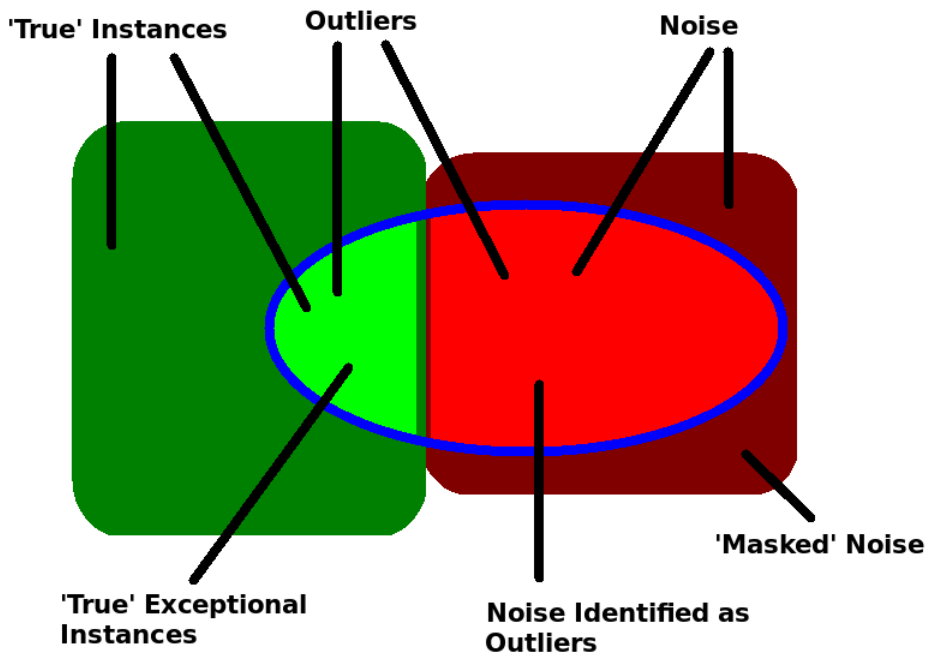
This thesis investigates data quality in software engineering data sets, with specific focus on accuracy of data. 'True' instances which are exceptional and appear as outliers are not seen as 'unwanted' since they contain valuable information about the problem domain. In order to resolve the conflation of the terms noise and outliers, they will be defined as follows:

- Outliers are instances with exceptional values in comparison with the rest of the data.
- Noisy instances are unwanted instances with inaccurate values.

The Venn diagram in Figure 1.1[2] shows the relationship of noise, outliers, 'true' instances and exceptional 'true' instances. Noise are data which are inaccurate. Since their creation differs fundamentally from 'true' instances, even if these 'true' instances are exceptional, they theoretically reflect this difference in creation in their values and therefore appear as outliers. Noted should also be another type of noise, marked as 'masked' noise in Figure 1.1. These are instances, which appear 'true', but which are inaccurate. This 'masking' can be due to two factors. Firstly, random values can coincidentally take on values which appear 'true'. This should be a seldom occurrence if the noise is 'truly' random. Secondly, noise can be fabricated intentionally such that it appears 'true'. In this thesis the focus is on random noise and not on systematic noise. As the intentional fabrication of 'masked' noise follows a system, it is considered beyond the scope of this thesis.

[2]Note that the diagram shows that the proportion of outliers identified as noise is higher than the proportion of 'true' exceptional instances. This is expected since 'true'-outlier instances are considered to be exceptional.

Figure 1.1: Noise, Exceptional Instances and Outliers



1.4 Class and Attribute Noise

Another aspect related to noise is whether the noisy values exist in dependent or independent variables. A number of researchers from the machine learning community [210, 181, 89, 205] extended the discussion about noise to two subcategories of noise; attribute noise (noise in the independent variables) and class noise (noise in the dependent variables). Zhu and Wu [210] stated that there has been substantial research carried out in the detection and handling of class noise, but attribute noise has been neglected. Zhu and Wu provided the following definitions for attribute and class noise: “The quality of the attributes indicates how well the attributes characterize instances for classification purpose; and the quality of the class labels represents whether the class of each instances is correctly assigned.” The class label, according to Zhu and Wu, is also a target concept which when performing classification is characterised by the attributes. Therefore it is an issues which is very much tied to the specific problem of learning a classifier, since attribute noise will hamper the classifier’s ability to predict a class variable.

The importance of the difference between attribute and class noise has to be tightly coupled to the “fitness for purpose” as well as the domain of the data, and is not always appropriate. In software effort analysis for instance, the target concept is measured effort. For the application of decision trees which assumes the classification of problems this target concept has to be discretised. These discretised values do still have to be considered as measured values since they are based on measured values. One could say that only the granularity is changed. It is not classified (or ‘assigned’) through the analysis of the remaining attributes. In fact, it is only one attribute which was chosen to be the class variable by the analysts.

The notion of a class label combines two aspects. Firstly, the class variable is a target concept with association to attributes describing an instance and their relationship to the class variable. In respect of effort analysis the target concept is chosen from a number of attributes of a data set. Each of the other attributes could be used as target concept if it would serve a purpose. That means that the quality of the “other” non-target attributes can be treated as class variable too and cleaned in the same way as the "original" class variable. In fact this is an approach Teng [179, 180, 181] followed when polishing data sets. Secondly, Zhu and Wu suggested a difference in the creation process of class values and attribute values, when they stated that class noise is likely to be cleaner since, as they explain with an example from the medical domain, the analyst will take more care to allocate a class to a set of attributes. This suggests that the class variable according to Zu and Whu is not measured, but allocated and due to being allocated, cleaner. Since class variables are not necessarily allocated and since they can be measured values too, the conclusion that class variables are cleaner is not necessarily true.

One of Zhu and Wu's conclusions based on a number of experiments states that attribute noise is usually "less harmful" than class noise. Brodley and Friedl [25] came to the same conclusion and subsequently focussed their work on class noise. Since class noise is considered more "harmful" the main focus of this thesis is class noise too. Also, since the data is measured software project data and class variables and attribute variables are interchangeable, this distinction between the two terms is not considered to be necessary for the discussion of noise in this thesis.

1.5 Research Goals and Thesis Objectives

This thesis aims to investigate data quality and specifically noise in software engineering data sets. Three data cleaning approaches, represented by one cleaning technique each, will be compared for their data cleaning effectiveness. Outliers and noise are often treated as one, but in this thesis the focus is on noise solely. Whilst the three approaches are usually used to deal with noise and outliers, the three representing techniques are tested for ability to deal with 'true' noise.

The following additional objectives were identified for this thesis. To:

- **Provide clarification and distinctions of the terms noise and outliers**
  The terms outliers and noise are often used interchangeably. Whilst they are overlapping, they are fundamentally different in creation and concept. This thesis provides clarification of the two terms [3].

- **Carry out a systematic literature review searching the empirical software engineering literature for evidence of data quality considerations**
  The work in this thesis highlights an important gap in empirical software engineering. This thesis will show to what extent data quality and noise have been dealt with in the empirical software engineering community.

- **Conduct a multi-pronged approach to investigate the effectiveness of three noise handling techniques**
  A single investigation is deemed insufficient to investigate the capabilities of noise handling techniques, since differentiating between noise and outliers is not trivial, and since the investigated noise cleaning techniques are derived from traditional noise handling techniques where noise and outliers are combined. Therefore a multi-pronged approach will be conducted to investigate the effectiveness of the three noise handling techniques. A multi-pronged approach is also beneficial in order to overcome shortcomings of a single investigation. Whilst the two investigations are based on a real

[3]This has largely be done in this chapter already.

world dataset where the ‘true’ noise level cannot be known, the third investigation is based on simulated data in order to have complete certainty about the underlying noise levels. A disadvantage of simulated data however is that it could be questioned how realistic it is. This is addressed by modelling the simulated dataset on a cleaned version of a real world dataset.

- **Highlight possible shortcomings of current automated noise handling techniques**
  Noise handling is only one part of a bigger data cleaning process. It will be shown that noise handling techniques should be integrated into a data cleaning process.

1.6 Thesis Structure

Chapter 2 describes data cleaning and noise handling in disciplines other than empirical software engineering. It shows that noise handling is only a subtask of the data cleaning process. It presents the three noise cleaning approaches, which are represented by three noise cleaning techniques tested in this thesis. Chapter 3 reports on a systematic literature review, focussing on how data quality has been dealt with in the empirical software engineering community.

Chapter 4 describes three noise cleaning techniques which are tested in this thesis.

Chapter 5 presents two investigations comparing the effectiveness of the previously presented noise cleaning techniques on the basis of the analysis of a real world data set. Since the true level of noise in the data set cannot be known, two separate proxy measures, namely predictive accuracy and the detection of implausible values, are applied to evaluate the three noise cleaning techniques. The studies presented in Chapter 5 highlighted limitations of analyses based on real world software engineering data sets. The subsequent study presented in Chapter 6 is based on simulated data sets modelled on a real world software engineering data set. Random noise was introduced in these simulated data sets and they were then subjected to the three noise cleaning techniques introduced in Chapter 4. This has the advantage that the noise cleaning techniques' abilities can be tested in an environment where the noise levels are known.

The thesis concludes with Chapter 7 providing a summary of the presented work, listing the contributions of this thesis. Limitations of the work with pointers to possible future work in the field of noise handling in empirical software engineering are listed too.

Chapter 2

Noise Detection and Noise Handling

2.1 Noise Detection and Noise Handling

Chapter 1 provided an introduction to the topic of data quality with a brief discussion about the importance of data quality considerations and the issue of noise. Since this thesis has its focus on data quality and data accuracy in software engineering data, the topic of how data quality has been addressed in research communities other than the empirical software engineering community with a special focus on noise handling needs to be discussed first. This chapter continues the data quality discussion from Chapter 1 by first addressing the data cleaning process. It then continues with a discussion of noise treatment options.

2.2 The Data Cleaning Process

Since data quality is defined as fitness of data for the purpose the data was intended for, the purpose will also influence data cleaning definitions. Maletic and Marcus stated in [116] and later in [117] that various definitions exist for data cleansing depending on the academic field they are used in. Whilst some areas like data warehousing connect the terms data cleansing and data cleaning with the merge/purge problem, where the focus is on the elimination of duplicate records resulting from merging two or more databases, other research areas connect the term with different data quality dimensions. For instance Müller and Freytag [131] classified data anomalies into syntactic, semantic and coverage anomalies, which are affected by the data quality dimensions completeness, validity, schema conformance, uniformity, density and uniqueness. These categorisations are reflected in the data cleansing approaches identified by Müller and Freytag. They listed parsing, data transformation (where data is transformed such that it adheres to a given format), integrity constraint enforcement, duplicate elimination and statistical methods. Only statistical methods deal with the type of noise that this thesis is concerned with.

Maletic and Marcus described data cleansing as an interactive procedure and state that "serious data cleansing involves decomposing and reassembling the data" [117]. Kimball [96] broke down this process of decomposition and reassemblance into six steps:

1. Elementising
2. Standardising
3. Verifying
4. Matching
5. Householding
6. Documenting

This means that data cleaning is a non-trivial process which not only consists of identifying problematic instances (Kimball's first four steps) and dealing with these instances (householding), but it also comprises the documentation of the data cleaning process and its results to allow replicability of the whole process.

Other variations [116, 131, 117] of the data cleaning process models exist, but the basic structure is the same:

- Define and determine error types [116, 131, 117] (Kimball's steps 1 and 2)
  For example values for the age of a person between 0 and 110 are considered plausible.

Values outside of this range are considered implausible.[1]

- Search and identify error instances [116, 131, 117] (Kimball's steps 3 and 4)
  Search for instances containing age values outside of the range identified in the previous step.
- Correct the uncovered errors [116, 131, 117] (Kimball's step 5)
  Delete or correct implausible-age instances.
- Post-processing [131] (Kimball's step 6)
  Document which instances were corrected or deleted.

Post-event data cleaning, cleaning already collected data, often focusses on outlier detection where outliers and noise are treated as unwanted [117]. This is likely connected with the fact that it is difficult to distinguish 'true' noise from outliers without domain knowledge. Maletic and Marcus stated also that exceptions are often indicators of the presence of noise. The problem with treating all outliers as noise can lead to the elimination of valid values which are not noisy [180]. However, outlier detection can be seen as a starting point for noise detection, especially if one follows Brodley and Friedl's assumption that noisy instances are "outliers in *any* model" [25].

Maletic and Marcus [117] classified outlier detection approaches into four categories: statistical, clustering, pattern-based and association rules. Statistical outlier detection utilises descriptive statistics such as the mean, the standard deviation and the range. These are used in conjunction with statistical confidence intervals. Although this approach is not considered very precise it is simple and fast. Clustering is a distance based outlier detection approach often using measures such as Euclidian distance. Pattern-based approaches attempt to establish a pattern in the majority of the data and instances not conforming with this pattern are seen as outliers or noise. Association rules establish associations between different attributes, therefore they also establish patterns. Instances not conforming with these patterns are seen as suspect and treated as noise [117].

[1]This step also ensure that domain knowledge is integrated into the data cleaning process.

As mentioned earlier data errors can be either eliminated or corrected. Teng [181] identified another outlier/noise handling approach[2] where errors can be ignored and left unchanged in the source. This approach would require the incorporation of robust algorithms, such that the influence of data errors can be limited.

The following two sections will discuss practical implementation of the three noise handling approaches.

2.3 Robust Algorithms and Filtering

The problem with ignoring noise and outliers is that the analyses of the data can lead to issues such as overfitting[3]. In order to avoid these problems robust algorithms need to be applied which avoid spurious results by reducing the influence of noise and outliers from the analyses [31]. According to Teng [181] the issue with ignoring noise and outliers is that they can still influence findings and therefore result in incorrect data analysis. That means a model is built and its complexity is reduced by ignoring outliers, but the outliers still have influence on the built model.

Gamberger and colleagues [58, 59, 60] used this principle for their saturation filter in order to filter out the ignored instances. Their saturation filter builds a model which is simplified when too complex, by eliminating instances leading to overfitting. They argued that their filtering algorithm avoids overfitting since the results of the noise elimination process will not influence the following analysis of the data and that therefore a hypothesis based on this analysis will not be influenced by the noise, thus combining robust algorithms and the filtering principle. In [60, 59] Gamberger and colleagues combined this saturation filter with a classification filter. Their tests on medical data sets indicate that all variations of the filters improve predictive accuracy. Saturation filter and the combination of saturation filter and

[2]Remember, in machine learning outliers equal noise. Whilst this is not sufficient for the work presented in this thesis it has to be recognised as an starting point for noise handling.

[3]Overfitting is a phenomenon where the results of an analysis describe relationships of a specific set of data, but these relationships are not transferable to other sets of data because they are spurious and therefore misleading in regards to the 'real' underlying relationships [134].

classification filter performed better than the classification filter by itself. They concluded that the combination of saturation and classification filter eliminated the instances which had the highest probability of being noisy. Their evaluation was based on two data sets from the medical domain with categorical dependent variables.

John [78] also utilised robust algorithms. A decision tree model is built which is simplified by pruning back over complex nodes. The difference between John's approach and the saturation filter approach by Gamberger et al. is that John repeats the pruning process until no more instances can be removed using this approach. He argued that the repetition of the process decreases the influence of noisy instances. He argued that excluding noise and outliers from the data set limits the impact of spurious findings but leads to information inefficiency since less data is available for the analysis. This approach was tested by John on 21 data sets from the UCI [136] database repositories. Whilst increasing predictive accuracy it reduced the size of the decision trees by on average 70%. Since reducing the size of a model (in John's case the decision trees) reduces the influence of data instances by ignoring them, instances are essentially eliminated after an initial model is built. This therefore reduces the data sets substantially too.

2.3.1 Filtering

Brodley and Friedl [25] noted that it is important to find algorithms which distinguish between noise and exceptional data values such that valid data does not get deleted during the noise elimination process. They compared a single algorithm filter, a consensus ensemble filter and a majority ensemble filter against each other. Brodley and Friedl stated that their approach differed from conventional outlier detection methods (the single algorithm filter) since it did not define outliers "relative to a particular model" as robust algorithms do. Their approach to noise detection is built on the premise that noise is independent of the data's underlying model. It combines a partitioning filter with an ensemble filter. A partitioning filter splits a data set into training set and test set. A data set is split into n parts creating n training sets each out of *n-1* parts of the original data set, as such producing n classifiers which test n test sets. A classifier "tags" an instance as noisy if the classifier predicted value for the dependent variable does not match the actual value.

Additionally Brodley and Friedl introduced the concept of multiple base classifiers for their ensemble filter. A base classifier is a noise detection algorithm. Brodley and Friedl used three base classifiers, a decision tree classifier, a nearest neighbour algorithm and a linear machine algorithm. For a consensus ensemble filter all of the base classifiers have to misclassify an instance to identify it as noisy. For a majority ensemble filter more than half of all of the base classifiers have to "tag" an instance as noisy. The single algorithm filter represented the traditional outlier detection mechanisms in their tests where training data and test data are the same. Brodley and Friedl evaluated the methods on five data sets with nominal and ordinal dependent variables. They concluded that the ensemble filters performed better in identifying noisy instances. The consensus ensemble filter was judged to retain most instances whilst eliminating less noisy instances. The majority filter eliminated more noisy instances, but also more 'true' instances.

Summarising two approaches of filtering can be applied. In both a model is built on a set of data and tests of misclassification or adherence to rules identify noisy instances. The two approaches differ by either keeping test set and training set the same (like robust algorithms), or by training a model on a different set of data and using it to test another set of data.

2.4 Correction of Instances

The third noise handling approach is noise correction which is also sometimes referred to as polishing. Correcting the imperfections has the advantage that more data for the analysis of the dataset will be available. It is based on two phases. The first step is identification of noisy instances. This is followed by the adjustment of the identified instances. This method was used by Teng [179, 180, 181]. Teng utilised a C4.5 decision tree for the identification of noisy instances by using misclassifications of instances utilising cross-validation similar to Brodley and Friedl [25]. The filtered data set was then used to build a new C4.5 tree model to predict new values for all attributes by switching each attribute from independent to dependent variable. Teng stated that the correction or polishing can successfully repair the data to some extend and therefore enhance the quality of the dataset. Teng evaluated the method on data sets with nominal or ordinal attributes only, and concluded that polishing was more successful in improving predictive accuracy than robust algorithms and predictive filtering.

2.5 Summary

This chapter discussed the issue of how data cleaning has been addressed in research communities other than the software engineering community with a special focus on noise handling and noise detection. It showed that the data cleaning process is a non-trivial process, and that noise handling and noise detection are subtasks of the overall data cleaning process. They have to be integrated into a data cleaning process to allow the input of domain knowledge and allow replicability of the data cleaning task. Noise detection identifies noisy instances by analysing a set of data and identifying instances which do not follow an established model. This chapter identified two different noise detection approaches:

- Training and test sets are the same, allowing the influence of identified instances.
- Training and test sets differ, thus reducing the influence of identified instances.

In both cases the identified instances are eliminated from the data sets. A third noise handling approach corrects instances rather than filtering them out, aiming to ensure information retention.

This chapter also showed that noise detection and noise handling is often combined with outlier detection and outlier handling. Since as shown noise and outliers are overlapping but fundamentally different, the focus in this thesis is purely on noise. There are many outlier detection mechanisms available, but their true noise handling capabilities are not tested. This is essentially the motivation for the work carried out for this thesis. Instead of proposing new algorithms, existing algorithms should be evaluated for their appropriateness for the different domains in empirical software engineering.

In this thesis the three approaches of noise handling will be tested for their adequacy for software engineering data. The three approaches are represented by three noise handling techniques, robust filtering, where training and test sets are the same, predictive filtering, where training and test sets are different, and filtering and polish, where noisy instances are corrected.

Before describing the three noise handling techniques in detail in Chapter 4 the following chapter will focus on the treatment of noise in the software engineering community.

Chapter 3

Noise Handling in Empirical Software Engineering

3.1 Introduction

Whilst Chapter 1 and 2 highlighted the importance of data quality for data analyses, defined the basic terminology needed in order to distinguish noise from outliers and examined the treatment of noise in research communities other than the empirical software engineering community, this chapter describes to what extent the empirical software engineering community has addressed data quality issues and specifically noise in software metrics data. Although data quality dimensions and issues might be recognised in empirical software engineering literature, the issue of noise in the raw data sets appear to be addressed sparsely. In their well known book Fenton and Pfleeger [51] list amongst other data quality dimensions the accuracy of data, but noise is only discussed as a difference between actual and predicted values (i.e. the goodness of a model's fit).

In order to assess the current state of affairs of data quality and noise handling in empirical software engineering a systematic literature review has been carried out. The objectives of this literature review are described in Section 3.2 and the method of literature identification is described in Section 3.3. Section 3.4 describes the findings with a more detailed examination of key papers relevant to the investigations of this thesis. Section 3.5 concludes this chapter summarising the state of affairs of data quality in empirical software engineering literature with connections drawn to the work presented in this thesis.

3.2 Objective for the Literature Review

The aim of this literature review is to identify *all* relevant studies and provide an overall and coherent picture of how data quality has been addressed in the empirical software engineering community. In order to produce a literature review as unbiased and repeatable as possible a systematic literature review [98] was carried out.

The main objective for the literature search was to discover which studies explicitly considered noise or data quality in empirical software engineering, and how these studies addressed this problem. Since the focus of this thesis is data accuracy, the focus of this literature review was also on accuracy (noisiness) as a measure for data quality, and not on other dimensions of data quality, described in Chapter 1. These dimensions might still have featured in some of the retrieved papers, but accuracy must have featured in some form too.

The following further sub-objectives were identified to aid the analysis of the results of the literature review:

- How significant do the community consider noise to be (in principle and in practice)?
- How do empirical analysts address this problem?
- Are there techniques that might be deployed to independently assess the quality of a given data set?

The results of the separate searches described in the next section were combined and then the retrieved articles were checked against the inclusion criteria. These criteria are that the article must:

- Focus on an empirical investigation of some aspect of software engineering or address some methodological issue relevant to such empirical research,
- Address data noise explicitly,
- Be refereed,
- Be written in English.

The next section describes the adopted method of identifying relevant literature.

3.3 Method of Identification of Relevant Literature

In order to survey the empirical software engineering literature concerning the subject of data quality, a systematic literature review was carried out. In recent years there has been increasing interest in establishing software engineering as an evidence based discipline and a crucial part of this process is the systematic review [98]. Systematic reviews are widely adopted in many other disciplines such as medicine, social policy, educational psychology, etc.. A systematic review is the process which requires an exhaustive scanning of all available literature that satisfies some agreed protocol that, amongst other things, will contain an unambiguous description of the inclusion criteria, which for the presented review were provided in Section 3.2, that a study must satisfy in order to be entered into the review.

The literature review consists of two parts. An initial literature search was carried out and the results of the subsequent analysis were presented to researchers of the software engineering community at PROMISE 2008 [109] where the audience, which included experienced researchers, expressed surprise. The feedback from the participants of the PROMISE 2008 workshop [65] resulted in a review of the initial search with special focus on the search terms. In order to improve the literature search the members of the program committee of PROMISE were contacted and asked to point at any of their publications or publications they were aware of which might fit the search objective listed in Section 3.2. The email used to ask the 23 program committee members is attached in Appendix A. Attached to the email was the list of papers found during the initial search. 11 replies were received of which five replies suggested 25 papers not retrieved by the initial search. 12 of these papers were deemed to conform to the search criteria. The relevant papers were then examined and suitable search terms were extracted. These search terms were then applied on several search engines.

Table 3.1 shows a comparison of the initial search strategy and the subsequent search strategy. It can be seen that the first search only used the search phrase "'data quality' AND software". This was deemed too limiting. The second search was adjusted such that search terms extracted from the PROMISE program committee papers were included. This was done to allow re-capture of the already captured papers. Some of these search terms may appear as very specific (i.e. “Issues Arising from the Data Collection” and “recorded with enough consistency and completeness”), but the method of extracting search terms from the papers was seen as a systematic approach. It also aimed to ensure that the PROMISE program committee papers were retrieved by the search. Whilst the initial search only utilised the bibliographic databases ScienceDirect, SCOPUS and IEEE Xplore, the second search additionally utilised the ACM and Springer databases. Note, Table 3.1 shows that in the initial search the journal Empirical Software Engineering was searched separately. This was not necessary in the subsequent search since it is published by Springer Verlag and since the Springer search engine was used.

Issues were encountered with the bibliographic search engines. Since the first search the interface of IEEE Xplore was changed, and the search results were not consistent with the first search. It is recognised that this is a problem for the validity of this literature review. To overcome this issue a name search was carried out to find additional publications of authors who resulted in a positive hit in the initial search and the second search.

Both searches omitted duplicate papers[1], which brought the final number of retrieved papers to 161. To reach this number hundreds of abstracts of papers retrieved from the bibliographic databases were scanned to determine if they concerned empirical software engineering applications. This was supplemented by an exhaustive, hand search of those sources considered to be particularly relevant, namely the conference series of ESEM, METRICS, ISESE, PROMISE and EASE[2]. It is recognised that the lack of online availability of published studies may slightly restrict the results of the literature review, however the author of this thesis is confident that the search has covered the major empirical software engineering publication venues. Therefore it is considered that the results provide an adequate view of the state of affairs in the empirical software engineering community.

After the search papers which were difficult to categorise were re-examined to clarify the

[1]Papers that report the same empirical study are only counted once using the most recent publication, e.g. [41, 82, 83] describe the same study and so only [83]

[2]Results from EASE proceedings were limited to 2008, 2007 and 2006 due to the lack of online availability of the remaining proceedings.

Table 3.1: Initial Search and Second Search Compared

	Initial Search	**Second Search**
Time	January 2008	January 2009
Search Terms	“data quality” and software	derived from terms found in papers from PROMISE PC - software AND (“noisy data” OR “data quality” OR noise OR “inconsistent pieces” OR “erroneous data” OR “clean the data” OR “data cleaning” OR “Issues Arising from the Data Collection” OR “recorded with enough consistency and completeness” OR “not very consistent”)
Bibliographic Databases	ScienceDirect, SCOPUS, IEEE Xplore	ACM, IEEE Xplore, ScienceDirect, Scopus, Springer
Conferences	ESEM, METRICS, ISESE, PROMISE and EASE (2006,2007)	ESEM, METRICS, ISESE, PROMISE and EASE (2006, 2007, 2008)
Additional Search	Journal Empirical Software Engineering	Name search of leading authors from found papers Inclusion of previous hits
Hits	23	161
Inclusion Criteria	written in English, published in peer reviewed publication, mentioning of data quality and noise according to our definition	

categorisation[3].

[3]The searches were carried out by G. Liebchen. The flagged papers were read and categorised by M. Shepperd (30%) and G. Liebchen (%70). If a paper's categorisation was difficult, both researchers read the paper and discussed the categorisation in order to find agreement.

3.4 Findings of the Systematic Literature Review

161 papers were considered relevant following a extensive literature review. This could be seen as surprising considering how many papers are written in the empirical software engineering domain every year[4], and considering that one of the inclusion criteria was that papers should simply address data quality in some respect. One would expect more papers at least mentioning some sort of data quality or noise consideration. A reference to all papers and their classifications for this literature analysis can be found in Appendix B in Table 7.3.

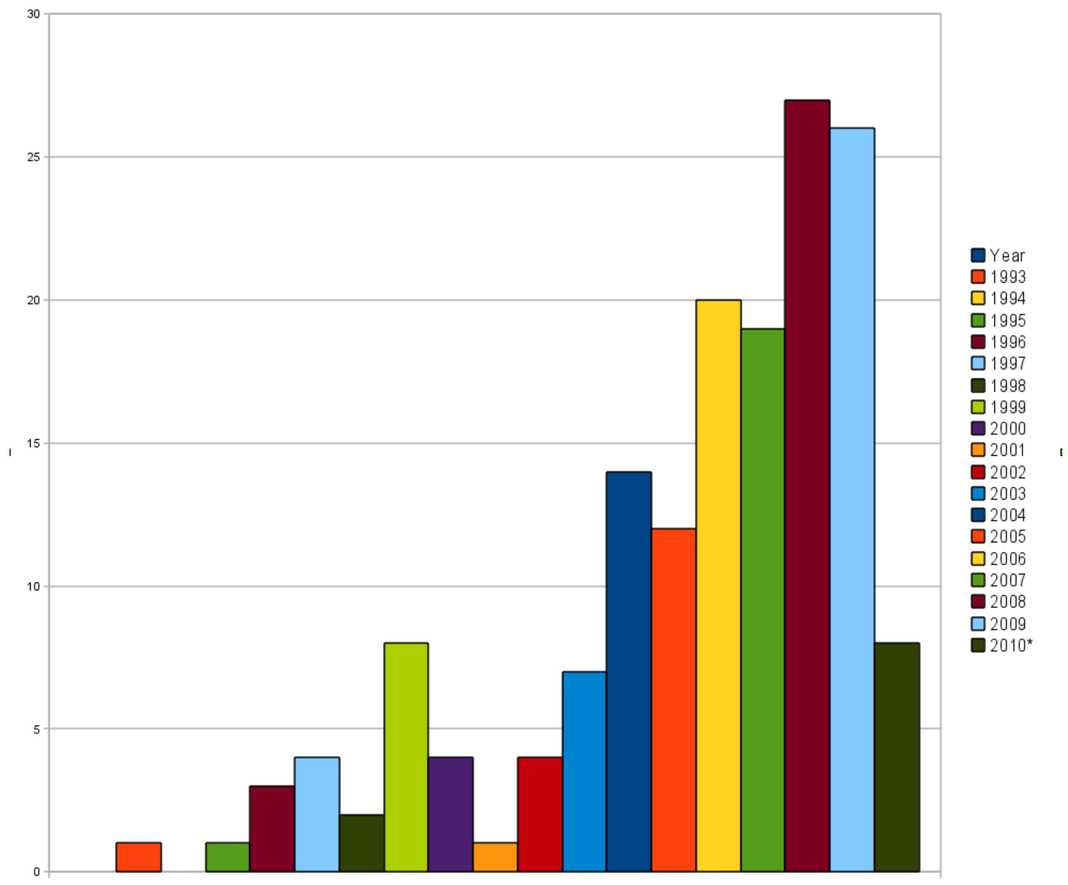
Figure 3.1 gives a breakdown of the 161 papers by year. Note that 2010 is incomplete. Nonetheless there seems something of an increase over time, suggesting that the community is giving the topic of data quality more explicit attention.

The earliest paper which the literature search could locate is by Li and Malaiya [106], who considered noise in the software quality domain in 1993. They state that noise is problematic for the prediction of software reliability growth models, and they applied smoothers to deal with this effect. Li and Malaiya's definition of noise falls into what is considered a mixture of outliers and noise in this thesis. The paper was included in the search nevertheless since its aim was to improve data quality.

Although a substantial majority of papers, 138 out of 161, considered data quality to be a threat to analysis of empirical data (see Table 7.3), not all papers agreed with this proposition. Indeed one author, Wesslén, suggested that the random, i.e. unbiased nature of noise meant that it could be ignored since presumably it would average out in the long run [199]. Whilst this may mean that measures of centre are largely unaffected there may be considerable impact upon the variance and the ability to fit and differentiate between predictors, and therefore noise can still pose a problem. Thus one cannot remain so sanguine.

[4]Note, that the literature review did not put any limits on the age of the papers. In order to get a feel for the number of papers written in the empirical software engineering domain, the search phrase "empirical AND "software engineering"' was entered in the basic searches of the ACM, IEEE Xplore, ScienceDirect, Scopus and Springer databases (data: 01/08/2010), resulting in the following counts of retrieved papers; ACM-6345, IEEE Xplore-1343, ScienceDirect-5103, Scopus-2382, Springer-2152. The total is in excess of 17000. Therefore only 161 out of 17000 (about 1%) papers written in the empirical software engineering literature explicitly discussed data quality.

Figure 3.1: Data Quality Papers Retrieved by Year



Wesslén also notes the possibility of intentional errors and states that these should be dealt with by validating the data, a sentiment hard to disagree with!

A small minority (14%) did not explicitly state if poor data quality could have a negative impact. In those cases Table 7.3 indicates this with a "?". Biffl and Gutjahr explain their trust in the quality of the data by referring to a quality data collection process and state that the remaining noise will not impact the results of their analysis since they are believed robust "enough". The concept of robust "enough" results is interesting, since the robustness should be able to be measured. This robustness could be tested by injecting noise and observing how robust the concept is at differing noise levels. This approach has been adopted by for instance by Berlin et al. [15] and Ahmed and Muzaffar [2].

The papers cover a range of topics within empirical software engineering. These include: meta-analysis, defect prediction and reliability modelling, effort prediction, PSP, reuse, change data analysis and studies evaluating different noise reduction strategies in software engineering datasets. Two of the domains were predominant, cost/effort prediction, which included productivity and schedule analysis, and software quality analysis, which included defects and defect predictions. Table 3.2 summarises the findings for these two domains of the papers. It shows the categorisations of the quality domain, the cost domain, the other remaining domains and the categorisations of all papers. 55 papers (34%) concerned the quality domain. 52 papers (32%) fell into the cost domain, and 60 papers (37%) were attributed to the "other" group. Some papers covered more than one domain. That explains why the counts do not equal the absolute total of 161. Four papers concerned both the software quality domain and the cost domain, of which one paper was found to cover all three groups, which was in fact the paper reporting about the initial literature search [109].

The majority of papers, 122 (76%), focussed on data quality of quantitative data, 17 papers (11%) were concerned with quantitative and qualitative data and 45 papers (28%) were concerned with data quality of qualitative data (e.g the papers by Li et al. [105] and Babar and Zhang [11] focused on data derived from interviews). An early paper by Johnson [80] highlighted the importance of quality review documents as a basis for software development reviews, but Johnson also pointed out the importance of accuracy of the metrics

Table 3.2: Paper Categorisation vs Domains

( ) = Paper in software quality domain and in cost prediction domain; ⟨ ⟩ = Paper in all domains;
{ } = Paper concerns qualitative and quantitative data;
n.a. = Paper did not state clearly if the concern was with quantitative/qualitative data.

	Quantitative		Qualitative		Data Collection	Manual Noise Checking	Automated Noise Checking	Empirical Analysis of Noise	Data Quality Meta Data	Total
	n.a.	Y	n.a.	Y	Y	Y	Y	Y	Y	Y
Software Quality	2 (1)	45 (3⟨1⟩) {6}	2 (1)	13 (1⟨1⟩) {6}	15 (1)	6	10 (1)	13 (1)	2 (1)	54 (4⟨1⟩)
Cost Prediction	3 (1)	48 (3⟨1⟩) {2}	3 (1)	3 (1⟨1⟩) {2}	10 (1)	9	3 (1)	5 (1)	15 (1)	52 (4⟨1⟩)
Other Domains	7	33 ⟨1⟩ {10}	7	31 ⟨1⟩ {10}	26	19	3	4	1	60 ⟨1⟩
All Papers	11	122 {17}	11	45 {17}	50	35	15	21	17	161

used in the review process. The data quality considerations raised in this paper were picked up by Lappalainen in [102] who added that quality issues may arise in PSP data because the data collection is one of two tasks software engineer have, and their need “to switch her context of thought from solving the problem (design, coding, etc.) to the recording of data” causes issues. Johnson and Zhang [79] only refer to data quality by citing [80].

11 papers did not specify whether the data quality concern was about quantitative or qualitative data.

3.4.1 How Was Poor Data Quality Dealt With?

After reporting what domains considered data quality and what data was subject of the data quality concerns, this subsection focusses on the approaches researchers have adopted to deal with poor data quality.

The results in table 3.2 indicate that the most favoured method (50 papers) of combating poor data quality is to avoid data quality problems through the data collections process. This was merely a suggestion in most papers which commended this course of action, since researchers in empirical software engineering most of the time are not included in the data collection process and therefore have to work with secondary data [66]. Frequently suggested was the use of automation of data collection processes and input validations, like range checking, in order to avoid data input errors. Combating data quality during the data collection phase could be characterised as a noise prevention activity.

Since as indicated above data analysis in empirical software engineering is often reliant on the use of historical software engineering datasets, noise prevention as data quality improvement will not be available for most researchers. This leads to researchers applying data cleaning procedures to data sets post event. Possibly due to the apparent ease of application the most prominent form of data cleaning is the manual data quality checking. Typically this involves increasing one’s confidence in a data set by some manual intervention such as independent scrutiny or the use of triangulation, for example measuring the same attribute in different ways or through inter-rater reliability analysis. This cleaning or scrubbing precedes the main analysis.

An example for this is [33] where automatically collected data was compared with manually collected data in order to determine how error prone manual data collection process is. The quality of the base line data quality of the automatically collected data was ensured by a visual inspection.

18 papers (11%) used data quality meta data in order to improve data quality. Some data sets, most notably the ISBSG project [75] effort and productivity benchmark data set, contain meta data that describe the perceived quality of each case or project. For ISBSG quality is graded between A (highest quality) and D (lowest quality). This is quite important for situations where organisations elect to contribute project data to a central repository and thus there is a reduction in control over collection procedures. In the situation of ISBSG the classification is principally guided by the completeness of a case, in other words high quality data are interpreted as possessing low levels of missingness, which could be misleading for analysts if they equate this classification to the level of noise in a data set. In this literature review the studies that utilised the ISBSG adopted the strategy of only using data graded as A or B. Note though that this does not accord with the view that noise concerns the difference between the "true" and recorded values for a data item. Indeed a complete case may contain many inaccuracies.

Two areas of data quality discussion in empirical software engineering which are directly related to the research presented in this thesis are the automated noise detection and the empirical analysis of noise prevalence. Table 3.2 shows that 16 papers (10%) carried out some sort of automated noise detection, and 21 papers (13%) carried out a empirical analysis of noise in software engineering data sets.

Next, papers which utilised automated noise checking are going to be discussed. After this, papers with empirical analysis of noise are going to be focussed on, followed by a discussion of paper which reported on automated noise checking combined with and empirical analysis of noise with the aim to assess the effectiveness of the noise checking procedures presented.

3.4.2 Automated Noise Checking

As mentioned in Chapter 2 automated noise checking is mostly carried out by utilising a machine learning algorithm to identify the noise.

Moses [130] presented a Bayesian probability model to assess the correctness of subjective categorisations of inter-modular cohesion. He used the cohesion classifications for six FORTRAN modules of 163 students to show that subjectivity could be inferred from agreement established using a Bayesian probability model. He states that this method could also possibly be used to quantify subjectivity in the field of cost/effort prediction. Moses also commented that no judgement about the quality of the student data could be made due to lack of information, but he suggested that the quality of the data might not have been the best due to interactions of the student raters.

Colombo et al. [32] also employed a Bayesian approach to deal with the impact of noise in software process data in order to avoid overfitting. The utilisation of a Bayesian network as robust algorithm has to be done since software process data “typically” contains noise. They investigated if class/switch events could be allocated to the developer who produced them. They analysed the log files of three Java programmers who produced about 10,000 class/switch events each. The results of their analysis showed that the Bayesian approach can be used to establish if a change was carried out by the same developer or by two different developers.

Rubin et al. [152] automatically eliminate incorrect change log entries by applying rules for correct change log entries in a software process mining tool (i.e. ignoring exceptional and infrequent change commits). Their results showed that their tool can be used to obtain process models and to analyse and verify some properties of software processes.

The remaining papers which utilised automated noise checking incorporated their investigations with empirical analyses of the data quality problem and will be discussed following the next subsection which only discusses papers carrying out empirical analysis of data quality without the employment of automated noise checking.

3.4.3 Empirical Analysis of the Data Quality

The topics of noise detection and handling are important, but their effectiveness can only be tested empirically. It is also important to consider the impact of poor data and its magnitude in software engineering data. In order to do this empirical investigations gathering and analysing data about data quality must be carried out. This is a non-trivial task since in many cases the ‘true’ value of a data item may be unknown. Thus unless the value is implausible in some way, the level of noise in a data set may be difficult to measure. However, an important example of data quality assessment, and an early paper on the topic, comes from Johnson and Disney [83]. They report that as part of the data recording process of the Personal Software Process (PSP) for 89 projects completed by ten participants they discovered 1539 primary errors. However, it must be stated that almost half (46%) of the errors were incorrect calculations and so can be addressed by the provision of better tool support. Another significant problem they encountered were missing data. There is a good deal of research on data imputation [112], but it is considered to be beyond the scope of this thesis. Overall Johnson and Disney concluded that poor PSP data could lead to misleading conclusions about the PSP itself, and in order to improve data quality manual data collection should be avoided and “external measures” should be used which can be interpreted for triangulation purposes.

At times the empirical analyses were rather limited. For instance, as mentioned earlier Counsell et al. [33] carried out a visual inspection of automatically and manually gathered data, and concluded that manual gathered data is error prone. Koru and Tian [99] included a question in their questionnaire for 119 contributors of open source projects where they enquire about the “consistency” of data in defect databases, when they carried out a survey about defect handling strategies in open source projects. The majority of contributors stated that their data was consistent, which appears not surprising since they were judging their own efforts.

In [184] Thomson and Holcombe inspected the change data repositories for 17 student-led software projects manually with the aid of comparison tools, and three types of errors were discovered. According to the authors type one error related to missingness or non-usage of the repository, type two error related to inaccurate data, and type three error related to errors which were due to “well known limitation” of the change data system used. Thomson and Holcombe warned in conclusion about the quality not only of data collected by students, but also of data in change data repositories in general.

Bachmann and Bernstein [12] investigated the data quality of software process data in change data repositories. The bug tracking data bases, which were also partially used for change requests, of five open source systems and one closed source system were checked for inconsistencies. First rules were applied to identify suspect instances which followed an additional manual inspection. Their quality measures included the rate of fixed bug reports, the rate of duplicate bug reports, the rate of invalid bug reports, and the rate of empty commit messages and the rate of linked bug reports. The closed source project showed lower values for duplicate bug reports and invalid bug reports than the open source projects (both about 1%). These lower values were due to the closed source project bug tracking database not being publicly available, and due to “well-planned” testing activities. Since the remaining data quality measures fell in the same range of the open source projects, the authors concluded that data quality differs “from project to project”, and that it was not possible to differentiate between open source or closed source data based on data quality alone. The bug reports in all databases were “badly” linked, and source code changes were untraceable “due to empty messages or missing bug report links”. Invalid and duplicate bug reports for open source projects varied between 12 and 34 % (duplicates) and 4 and 34 % (invalid reports), which indicates poor quality bug report information.

The investigations by Mendes[120] and Mendes et al. [123] were both based on the same data when investigating cost estimations for web projects. According to the authors data quality of effort data was ensured by asking the companies which collected the data how the data was collected. 83% of the projects based their effort data on more than just guestimates indicating sound data. Whether this fact can really be seen as an indicator for good data quality appears arguable.

In [92] Khoshgoftaar et al. investigated the impact of noise on missing data imputation in software quality data. They used a large military command, control, and communications system data set (CCCS) to produce four data sets with varying noise levels. Two of the produced data sets contained inherent noise, noise already in the data set for the outset, and artificial noise. One data set only contained inherent noise, and one according to the authors did not contain any noise. This is interesting considering the statement by De Vaux and Hand that the absence of noise cannot be proven [37]. Koshgoftaar et al. reported that the noise free data set was achieved with the help of "expert input and several [unspecified] data analysis procedures". The authors concluded that regardless of the imputation method used, "ignoring data quality can lead to erroneous conclusions".

In [190] Van Hulse and Khoshgoftaar investigated the impact of noise on 11 machine learning algorithms using five NASA software project data sets containing fault data and two data sets from the UCI repository [136]. They injected differing levels of artificial noise and concluded that noise has an adverse effect on the learners' abilities to predict unseen instances. Their results showed that simple algorithms like naïve Bayes and nearest neighbour algorithms performed better in noisy environments than more complex algorithms like support vector machines and random forest algorithms. They also tested seven data sampling techniques on the noisy data sets again concluding that simpler sampling techniques were more effective.

The previous paragraphs showed that some researchers attempted to quantify the extent of poor data quality in software engineering data sets. Some of these investigations were very limited and only part of the main investigations. The investigations indicate that noise problems can be assessed by means of proxy measures and verifications like expert judgements and predictive accuracy of learners. These measures could then be used to investigate the effectiveness of data cleaning techniques. Automated noise checking techniques have the benefit that they can make the data cleaning process less onerous to analysts, and they do not need costly manual interventions, but their effectiveness needs to be proven in order to prove their true value to the person charged with the data cleaning task.

3.4.4 Automated Noise Handling with Empirical Analysis of the Data Quality

Apart from the two already mentioned papers by Khoshgoftaar and colleagues, in which the authors carry out an empirical investigation into the impact of noise, Khoshgoftaar and colleagues provided most papers combining automated noise handling with an empirical analysis of data quality, thus attempting to prove the value of their different automated noise cleaning techniques. As with the papers above their investigations utilise software quality data sets with nominal or ordinal dependent variables.

In [209] Zhong et al. used a clustering- and expert-based software quality estimation method, where instances are first clustered using both a k-means algorithm[5] or a Neural-Gas[6] algorithm in comparison followed by expert inspection of the results. The authors tested their approach on a data set provided by NASA containing software quality metrics for 520 modules. They suspected that the data set contained noise. They compared the effectiveness of their method in identifying noise by comparing instances which were considered noisy by their proposed method with instances identified as noisy by an ensemble classification filter. Their results show a high degree of consistency. This only indicates though that the ensemble filter and their technique identified the same instances. It does not really show if the instances were noisy or not.

In [86] Khoshgoftaar and Rebours compared different settings of an ensemble partitioning filter resulting in the comparison of a classification filter, an ensemble filter, a multiple-partitioning filter and an iterative partitioning filter when filtering out instances identified as noise. Again a NASA data set was used which contained originally data about the faultiness of 10883 software modules. After removing inconsistent modules, modules with identical software metrics but different class labels (faulty/ not faulty), 8850 modules remained. The resulting data set was named JM1-8850. The different filters were applied and their agree-

[5]The k-mean clustering algorithm groups a number of instances (observations) by attributing them to the k^{th} group (cluster) with the nearest centroid [115].

[6]Similar to the k-means clustering the Neural-Gas clustering algorithm compares the distance between observations and clusters, but it also revises the distance following several iterations [119].

ments were compared. According to the authors the classification filter was the most aggressive, removing most instances, and the ensemble filter removed the least instances. Whilst retaining of good instances should be a good outcome it is not clear if the identified instances were noisy or not. Seemingly addressing this point the authors stated that an agreement of all filters “provides a good indicator of the true quality of the instances”. This statement appears unproven.

The NASA data set JM1-8850 was used in [178] by Tang and Khoshgoftaar together with another NASA software quality data set, named KC2 and which they left as is, in order to evaluate a k-means based noise detection technique. The authors tested the noise detection capabilities of a k-means outlier detection method by comparing the predictive accuracy of models built from the cleaned data. The authors noted that outliers are not equal to noise, but state that a subtask of outlier detection methods is the detection of noise. They stated that after applying their technique the predictive accuracy increased significantly.

In [91] the NASA data set JM1-8850 was cleaned further using clustering and predictive filtering resulting in a data set with 2445 instances, named JM1-2445, which was said to be “noise-free” by Khoshgoftaar et al. . This cleanliness was verified with “perfect 10-fold cross-validation classification results of a C4.5 classifier”. It seems questionable if this true “noise-free status” could really be verified in such way, since it only indicates how well the data predicts the instances contained in itself. JM1-2445 was then used to evaluate a rule-based noise detection technique. Noise was artificially introduced and the ‘effectiveness’ and ‘efficiency’ of the technique were compared against a C4.5-based classification filter. The authors defined ‘effectiveness’ as “the number of instances correctly predicted as noise divided by the total number of noisy instances in the data set”. ‘Efficiency’ was define as the number of actual noisy instances in the number of predicted as noisy divided by the total number of instances predicted. This is essentially gratuitous new terminology for precision and recall. Their results indicated that their proposed method showed higher ‘efficiency’ and higher ‘effectiveness’ compared against the classification filter when the number of attributes injected with noise increased. Also the performance of their technique “did not seem” to be impacted by increased noise levels (increased artificial noise levels from 10% to 20%) as opposed to the classification filter.

The "clean" data set JM1-2445 was used by Khoshgoftaar and Van Hulse as basis for the investigation in [89], but 418 "known" noise instances from JM1-8850 were left in place. The noisiness of the 418 instances was verified by software metrics expert judgement. The resulting data set JM1-2863 was used to evaluate their method "Attribute of Interest" (AOI), which is a methodology where the noisiness of a variable is determined by making it the dependent variable and applying their noise ranking technique PANDA (Pairwise Attribute Noise Detection Algorithm). PANDA ranks instances according to noisiness summing the combination of a distance measure extracted from the mean and standard deviation of a distribution, which the authors called "noise factor". The authors compared their methodology against a classification filter and a ensemble filter, and their results indicated that their method AOI showed higher effectiveness and efficiency as defined in the previous paragraph. Ranking instances according to noisiness is an interesting concept and potentially very useful. The issue of setting boundary values, which ranks are considered noisy, is difficult, especially if there are no clues about the level of noise in a data set.

PANDA's noise detection capabilities were also investigated by Van Hulse et al. in [191], this time without the incorporation of PANDA into AOI. PANDA was compared against a nearest neighbour distance-based outlier detection technique. This technique was chosen by the authors because like PANDA it does not rely on the incorporation of a class variable (see the discussion of class versus attribute noise in Chapter 1). Both techniques produced a ranking of instances from a data set according to the likelihood of the presence of noise. A software engineering expert then compared both lists categorising each instance independently. Two data sets were used to compare the two techniques. JM1-2445 which was considered "clean" in [91] and [89] was subjected to the two techniques. The 250 highest ranked instances where then inspected by a software metrics expert in order to distinguish between noise, outlier and exception. Outliers according to the authors are instances, which appear "inconsistent with the remainder of a set of data, or which deviates so much from other observations so as to arouse suspicions that it was generated by different mechanism." Noisy instances according to Hulse et al. are instances with one or more attributes "corrupted or incorrect relative to the values of the other attributes". Hulse et al. defined exceptions as instances, which appear noisy due to attributes in an instance, which do not follow the general distribution in comparison to other instances and their attributes in the data set. It appears unclear what Hulse et al. recognise as the difference between outliers and exceptions. It is not described how the expert distinguished between the three problematic categories of instances. The second data set used to compare the two techniques was the CCCS data set which did not undergo any preprocessing. This time the 30 noisiest, highest ranked, instances were inspected by the software metrics expert. In conclusion the authors stated that PANDA identified more noisy instances, fewer outliers and fewer "typical" (clean) instances than the nearest neighbour outlier detection technique. The employment of a software metrics expert for the identification of noisy instances is interesting, but the lack of description of the expert's noise identification process does not aid replicability of the evaluation process.

In [192] Van Hulse et al. investigated noise correction using Bayesian multiple imputation. The CCCS data set, which contained noise, was again inspected by a software metrics expert. 20 'unmistakably' noisy instances were identified and the expert calculated corrected values for the inherent noise. The author's technique is compared against a linear-regression based technique for correcting the values of a continuous dependent variable (number of faults in a module). Unsurprisingly the author's technique is more accurate than the simple linear-regression based correction of noisy instances.

The main focus of Seliya and Khoshgoftaar in [157] was the software quality estimation accuracy of an ensemble filter. A ensemble filter combines the outcomes of several classifiers with a majority vote, to clean noise. The authors used the JM1-8850 data set to investigate the effectiveness of missing value imputation for unlabelled instances, but they also investigated the noisiness of these unlabelled instances, suggesting that the reason they were unlabelled was an indicator for poor data quality. They stated that about half of the unlabelled instances were identified as noisy by the ensemble filter. Their definition of noise follows a machine learning paradigm where noise and outliers are combined. The effectiveness of the ensemble filter as noise detection technique was not investigated.

In [87] Koshgoftaar and Rebours compared the data cleansing abilities of two versions of a partitioning filter; multiple-partitioning filter and iterative-partitioning filter. Again JM1-8850 was used as a basis for their investigation. The predictive accuracy of a model built from the clean data was utilised as indicator of the effectiveness in noise elimination of the used filters. Their conclusions showed that number of cross-validations and the number of agreements of different filters can significantly impact on the predictive accuracy of models built from cleaned data.

In all their papers Khoshgoftaar and colleagues refer to class and attribute noise, but this issue was discussed already in Chapter 1 and deemed as not appropriate for the purposes of this thesis[7].

Comparing the counts of papers which focussed on automated noise checking and the empirical analysis analysis of data quality across domains, it can be seen that the strongest represented domain is the software quality domain. This is due to the efforts of Khoshgoftaar and colleagues who found a niche for their research. In the effort prediction domain only three papers reported on empirical analyses of data quality, of which two were by the author of this thesis and his colleagues [110, 111], and one paper which commented on the work presented in this thesis [205]. Only five papers in the effort prediction domain discussed automated noise checking techniques. Again the three papers which reported on empirical analyses are included in this count. The two remaining papers were by Mendes and colleagues [120, 123], and incorporated interview responses about the quality of data, but their conclusions drawn from these responses were likely to be overoptimistic. Research into self-relevant feedback and estimation has indicated a tendency to overoptimism [26, 161]. It is also questionable if data would likely to be criticised by the people who build their estimations and predictions on this data, since poor data quality would devalue these estimations and predictions.

None of the papers retrieved by the literature search described datasets before and after

[7]Reasons for focus on Class noise in this thesis:

- Class noise is seen as more harmful than attribute noise.
- A attribute variable can be swapped with the class variable, making it the class variable in order to consider noise in any variable.

data cleaning, listing all as noisy identified instances. This lack of documentation in form of protocols of the preprocessing of data sets is clearly an issue since it reduces replicability of studies.

The discussion of four papers which focussed on noise in empirical software engineering data sets is postponed until later. Their categorisations are included in the counts presented in Tables 3.2 and 7.3. Papers [108, 110, 111, 109] reported on investigations which are presented in this thesis. The fourth paper is a recent paper by Yoon and Bae [205] which will be discussed in Chapter 7 since it cites and builds upon some of the work presented in this thesis.

3.5 Concluding Remarks about Data quality in Empirical Software Engineering Literature

This chapter presented a systematic literature review investigating how the empirical software engineering community has addressed data quality issues with a specific focus on noise.

The literature search retrieved 161 papers, which appears a relatively low number considering that the papers only had to mention noise or data quality without actually proposing actions to tackle the issue, and also given the number of papers written in the empirical software domain every year. This means that the issue of data quality is somewhat neglected in a domain where the most important input is data. Poor data quality clearly poses a threat to the validity of conclusions from analyses, and it is questionable if poor quality data sets should be used if the ability to produce meaningful conclusions is limited, if not impossible.

Whilst the majority of the retrieved papers considered poor data quality an issue, very little has been done to combat poor data quality. The majority of papers suggested improvement of data quality during the data collection phase, recommending preventative techniques such as appropriate tool support for data collection. This is not always possible since analysts of empirical software engineering data are often limited to the analysis of historical data only [66].

No evidence has been found of the application of data quality protocols which describe what preprocessing steps have been taken to improve the quality of a data set, and which instances have been identified as noisy.

The dominant practical approach to deal with poor quality of historical data is manual noise checking. Data quality meta-data has also been used for the elimination of poor data quality instances from data sets, but unfortunately these data quality meta-data tend to be surrogates for missingness only.

Presently there appears to be little work available, which independently assesses the quality of a given data set. Researchers who investigated automated noise handling techniques did not always make their evaluations transparent and partly based their work on 'clean' real world data sets, which is a rare occurrence. 'New' techniques were introduced, but their data cleaning abilities were tested only by utilising unproven proxy measures. Since the 'true' extent of noise cannot be known [37] the use of one proxy measure alone to test a technique's 'true' data cleaning ability appears insufficient.

Automated noise checking and handling are still at a relatively early stage, and the effectiveness of the proposed methods in identifying and handling of noise seem not proven. Investigations which provide a multi-angled approach for the prove of effectiveness should be employed without fear of failure of the investigated technique.

The lack of work investigating independent means for assessing the quality of empirical software engineering data is another motivation of the research presented in this thesis. Instead of creating new algorithms and testing them with measures which appear to have little meaning for data quality in software engineering data sets, existing algorithms ought to be tested for their appropriateness for software engineering data.

In this thesis three noise handling approaches are compared to each other in the form of three techniques each representing one approach. Their data cleaning effectiveness is compared in the software engineering domain. More specifically the three techniques are compared for their data cleaning abilities of software project effort data.

The next chapter presents the three techniques, robust filtering, predictive filtering and filtering and polish.

Chapter 4

Robust Filtering, Predictive Filtering and Filtering and Polish

4.1 Introduction

The results from the systematic literature review presented in the previous chapter show that data quality has been somewhat neglected in empirical software engineering. In particular automated noise checking and empirical investigations into the effectiveness of automated noise checking techniques are lacking attention.

In this chapter three noise handling techniques will be described, and their data cleaning capabilities will be evaluated in the following chapters of this thesis. The techniques are robust filtering, predictive filtering, and filtering and polish, each representing one of the approaches for dealing with noise. Filtering where the training set and test set are the same are represented by the robust filtering technique. It builds a classifier which is pruned in order to avoid overfitting. The pruned instances are filtered from the data set, but subsequent models still are partially influenced by them. The filtering where training set and test set differ are represented by the predictive filtering technique. It predicts values of a class (dependent) variable for unseen instances which are then compared with the actual values. Mismatches are considered as noise and deleted from the data set. Corrections of instances are represented by the filtering and polish technique which uses the predictive filtering technique as initial step. A classifier is built using the remaining instances left after predictive filtering. This classifier is used to predict and alter values for the instances filtered out during the predictive filtering procedure leaving a 'polished' data set. All three techniques utilise classification decision trees as the classifier. Decision trees were chosen as the basis for all three techniques since they are widely adopted and are relatively simple to use. The terms classification tree and classifier are from now on used interchangeably in this chapter.

The chapter continues with a brief discussion about decision trees, followed by a description of the three techniques.

4.2 Decision Trees

The three techniques presented in this chapter utilise decision trees in the form of the CART (Classification and Regression Trees) algorithm as introduced by Breiman, Friedman and Ohlsen and implemented in the RPART package [183]. CART is a trademark and the RPART package is a freely available version for R and S-Plus. Decision trees are simple to use and have been widely and successfully applied in machine learning in order to build models of data relationships [179, 180, 181, 25, 87]. Decision trees are popular since they are easily understood by humans. They can build models without the assumption of underlying relationships of the included attributes.

A decision tree is a tool which allows prediction of target values on the basis of mapped observations and subsequent conclusions about these observation. Decision trees come in two variations, as classification trees which predicted discrete outcomes, and as regression trees which can predict continuous outcomes. A decision tree can be thought as a sequence of questions, which leads to a final outcome. Each question depends on the previous question which is represented as a branch in the tree.

The two most commonly used decision tree algorithms are the C4.5 and the CART builders, which differ in the number of nodes after each split. Whilst the CART algorithm is binary based meaning that every node must result in two branches the C4.5 algorithm is not necessarily binary based. As mentioned above all three data cleaning techniques utilise an implementation of a CART decision tree.

Decision trees are robust and can deal with problems like missing values. The CART algorithm does this by applying a surrogate test in order to approximate the outcome. That is, examining the combination of the non-missing values in an instance with the missing value and searching for similar combinations in other instances in order to approximate the missing values.

4.3 Decision Trees for The Three Data Cleaning Techniques

For the investigations presented in this thesis a classification tree is constructed using the CART algorithm. A subset of a given data set is used to train the classification tree. Since the application domain is effort prediction, effort is chosen as class variable (dependent variable). For this purpose effort was discretised. Discretisation is necessary for the purpose of comparison. It can help to distinguish if a value is correctly classified or not. As mentioned above decision tree algorithms can also be used to create regression trees, but this complicates the classification of noisy and non-noisy instances. A regression tree could also be used to achieve noise classification by creating intervals, where a value within a range from a guide value is considered as not noisy and a value out of that range is considered as noisy. The difficulty here lies at determining the size of the mentioned range or interval. For instance a value y is the actual value of a class variable in a data set, and the regression tree predicts the value $\hat{y}$ for this class variable. How big has the interval n to be, when $\hat{y}$ - $n \geq y \leq \hat{y}$+n, to consider a value as noisy?

The experiments presented in this thesis utilise the discretisation of the class variable. The discrete values are used as categories, where a misclassified instance can be seen as noisy. Whilst the first study, presented in Chapter 5.3 only utilised five bins[1] for these categories, the subsequent investigations use 10 bins. This was done because a lower number

[1]The boundaries of each bin was calculated by creating percentiles from minimum and maximum values.

of bins decreases the likelihood of misclassification since the classification categories will be too coarse. Another issue with a small number of bins becomes apparent when predicting values using a model based on the given bins. Since, as mentioned in the first example, only five bins were available, only five different values can be predicted. This issue will be discussed in more detail in the Section 4.6.

All noise handling techniques described in this chapter utilise cross-validation. Cross-validation is a standard procedure in machine learning which aims to test a model on an independent by equivalent dataset to the dataset a model is built from. It comprises splitting the dataset into *k* folds, using *k-1* folds to build, or train the model and one fold to test the model. All techniques split the dataset into five folds where 80% of the data are used to train the trees and, in the case of predictive filtering, 20% to test. For this purpose the data set needs to be split in equal parts such that each part can be used as part of the training set or as test set. These parts (folds) were non-overlapping, and the order of all instances in the dataset was randomised. As mentioned above, for the purpose of classification the chosen response variable needs to be discretised. Both steps, the splitting of the dataset and the discretisation are described in Table 4.1 which shows an initialising algorithm.

The following sections will describe the noise handling techniques evaluated in this thesis in more detail, starting with robust filtering, followed by predictive filtering, and the filtering and polish technique.

Table 4.1: Algorithm for the preparation of a data set for all noise handling techniques

Algorithm for the preparation of a data set for all noise handling techniques

1. Discretise the response variable by creating bins according to deciles.
2. Split data set DSO into 5 equal parts, DS1, DS2, DS3, DS4 and DS5.
3. Create overlapping subsets of DSO
   (a) Create TS1 by combining DS1, DS2, DS3 and DS4.
   (b) Create TS2 by combining DS2, DS3, DS4 and DS5.
   (c) Create TS3 by combining DS1, DS3, DS4 and DS5.
   (d) Create TS4 by combining DS1, DS2, DS4 and DS5.
   (e) Create TS5 by combining DS1, DS2, DS3 and DS5.

4.4 Robust Filtering

Robust filtering, sometimes referred to as pruning, is essentially a robust algorithm avoiding overfitting by utilising a pruning strategy to reduce the size of a decision tree. This can be used as a method to detect outliers, however in this thesis its noise detection effectiveness is investigated. Chapter 2 has shown that the issues of outliers and noisy instances are very closely related. Instances with random noise are often discovered due to the fact that they can exhibit characteristics which are different to the rest of the instances in a data set. Outliers are instances which are also exceptional, but ’true’ in so far that they are caused by exceptional circumstances rather than errors. Since noise and outliers are so difficult to distinguish they are treated as the same by a large part of the machine learning community. In this thesis the focus is noise in data sets in the domain of software effort prediction. Therefore the noise detection and cleansing properties of the presented techniques are investigated rather than the ability of discovering outliers, which can be done with conventional outlier detection methods.

The robust filtering technique is sometimes referred to as a robust algorithm since a tree is built using all instances, but which is reduced after analysing the tree’s properties. This differs to other filtering methods in that the pruned, or filtered, instances from the robust algorithm are used to build the first model which is the basis for the subsequent ‘pruned’ or cleaned model. The training set is essentially the same that is filtered. In the other filtering approach presented in this chapter the identified noisy instances are filtered out before a model is built, and these instances therefore have less influence on the new ‘clean’ model. Robust filtering is still a filtering method since it singles out instances leading to overfitting, which are then excluded from subsequent analysis. John [78] utilises the pruning strategy to clean a data set by iterating the pruning technique, and argues that the influence of noise on the final model will be reduced due to the iteration of the pruning technique. The robust algorithm used in the investigations presented in this thesis repeats the pruning procedure for all combinations of *n-1* parts of a *n* split, where *n* is the number of cross validation folds. Pruning is not continued after this for two reasons. Firstly all three techniques were kept as basic as possible in order to allow the comparison of robust algorithms, predictive filtering and filtering and polish. Secondly each pruning iteration results in the elimination of instances from a data set, resulting in information loss. In order to minimise this information loss the decision trees will only be pruned *n* times.

An *n* fold cross validation is carried out by splitting the data set into *n*, five in case of the investigations presented in this thesis, parts. This step is described in Table 4.1. The next step is to combine *n-1* parts in order to create *n* overlapping parts of the data set.

A tree model is built, utilising the RPART tree builder, of each of the overlapping subsets. These tree models are pruned utilising the 1-standard error (1-SE) rule, which incorporates Occam's razor where a simpler model is to be preferred to a complicated model. This is essentially an implementation of the saturation filter proposed by Gamberger and colleagues [58, 59, 60]. Occam's razor is realised by inspecting each split in the tree focussing on the cross-validation error. The aim is to find the best number of splits which takes the smallest cross validation error (xerror) adding the corresponding standard error (xstd). According to the 1-SE rule the complexity parameter (CP) for the least overfitted tree is established by ensuring that xerror + xstd of the most complex model is smaller than the xerror for all valid splits, so preventing an overly complex tree and avoiding overfitting. Instances which are in the pruned branches are removed from subsequent models and analysis.

The method is explained by the following example. A tree is built and the complexity parameter table (Table 4.2) showing a summary of the overall fit of the model is produced. According to the 1-SE rule, xerror and xstd are added together, resulting in 0.866927. Now this number is used to inspect the xerror column. It can be seen that xerror is smaller for all split steps 5 - 6. Therefore the tree will be pruned setting the CP value between 0.034483 and 0.045977. A new tree is built by stopping the growth of a tree after the CP fell below 0.045977. This new tree is pruned.

By comparing the unpruned tree in Figure 4.1 with the pruned tree in Figure 4.2, the pruned nodes can be identified. Figure 4.3 shows which nodes are pruned, resulting in the exclusion of the corresponding instances from the data set.

The steps for the robust filtering are explained in the algorithm in Table 4.3.

Table 4.2: Complexity Parameter Table

Split Step	CP	nsplit	rel error	xerror	xstd
1	0.126437	0	1.00000	1.11494	0.011435
2	0.114943	1	0.87356	1.09195	0.019602
3	0.091954	2	0.75862	1.02299	0.032861
4	0.045977	3	0.66667	0.87356	0.047477
5	0.034483	4	0.62069	0.79310	0.051939
6	0.022989	5	0.58621	0.80460	0.051404
7	0.011494	7	0.54023	0.81609	0.050837
8	0.010000	8	0.52874	0.81609	0.050837

Figure 4.1: Unpruned Tree

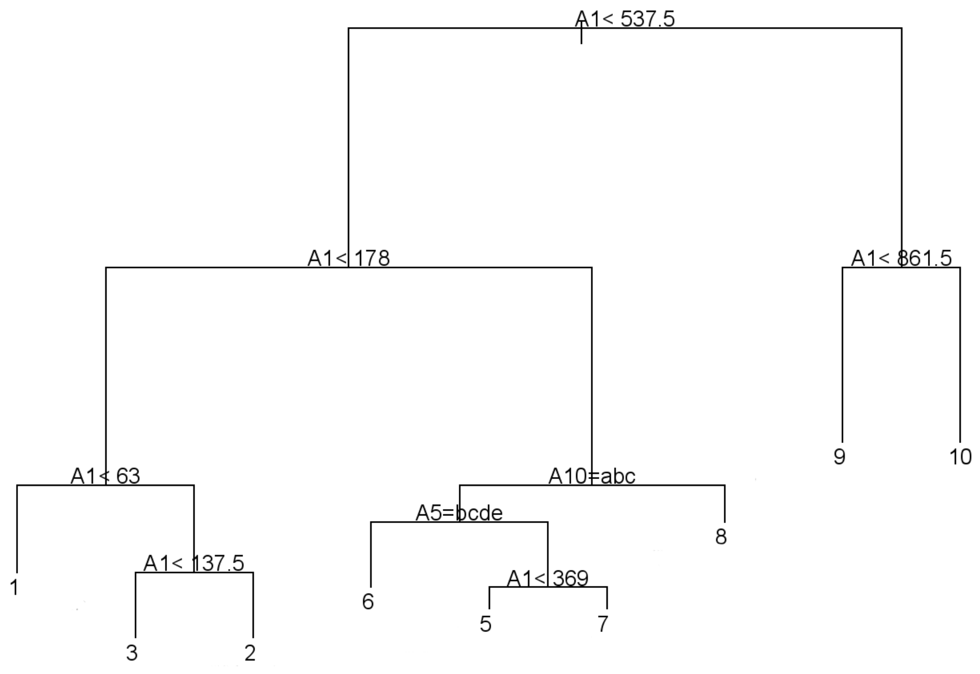


Figure 4.2: Pruned Tree

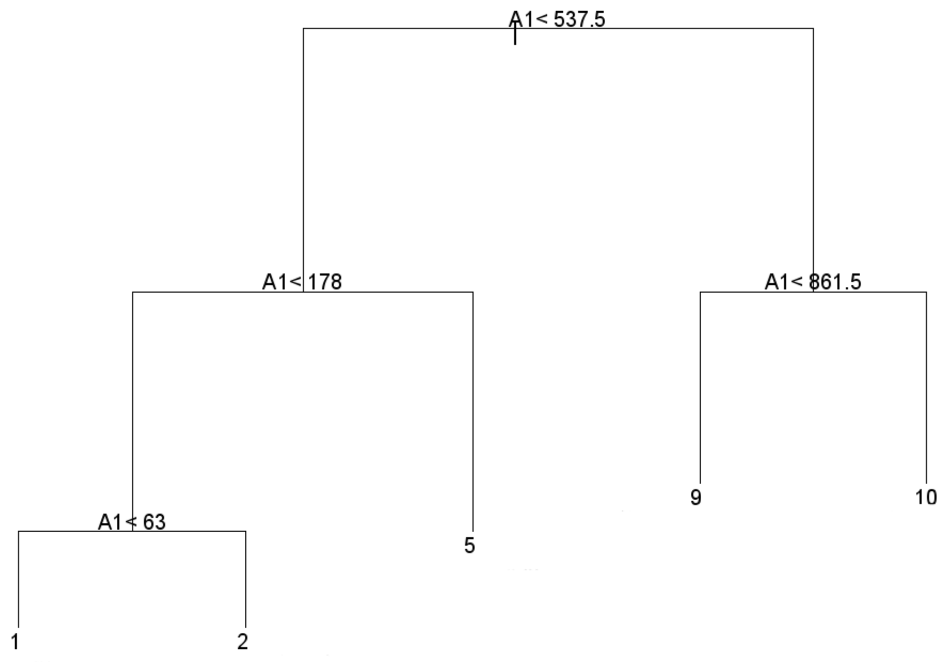


Figure 4.3: Pruning of a Tree

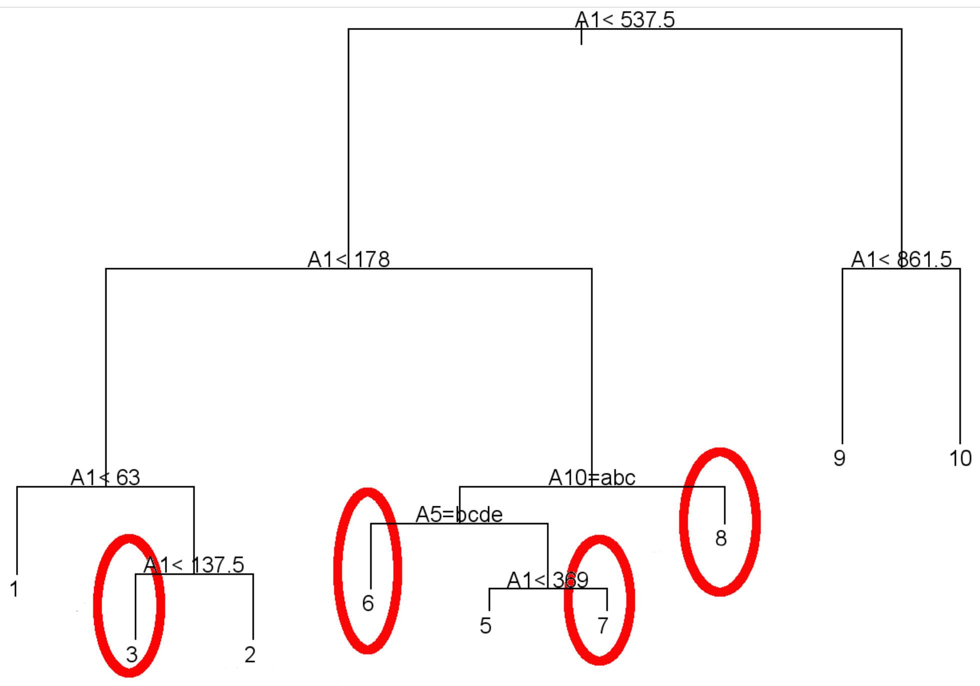


Table 4.3: Algorithm for robust filtering procedure

Algorithm for robust filtering procedure

1. Build classification trees.
   - Build classification trees TS1_Tree to TS5_Tree for each training set TS1 to TS5.
2. Prune the trees
   - Prune each tree TS1_Tree to TS5_Tree by applying the 1SE-approach creating new trees TS1_Tree_Pruned to TS5_Tree_Pruned.
3. Identify instances leading to overfitting.
   - Compare each unpruned tree (TS1_Tree to TS5_Tree) with its pruned version (TS1_Tree_Pruned to TS5_Tree_Pruned), identify pruned leaf nodes and related instances in data set TS1 to TS5 creating data set TS1P to TS5P.
4. Combine TS1P, TS2P, TS3P, TS4P and TS5P. Each instance in the data sets is identified and highlighted as noisy or not-noisy. This results in 5 additional boolean values for each instance TS1_noisy, TS2_noisy, TS3_noisy, TS4_noisy and TS5_noisy which indicate if an instance was identified to be noisy in any of the data sets TS1P, TS2P, TS3P, TS4P and TS5P. The result is data set DSP.
5. DSP is split in DSP_clean and DSP_noise. DSP_noise contains all instances which hold "true" in four of the values for TS1_noisy, TS2_noisy, TS3_noisy, TS4_noisy and TS5_noisy. DSP_noise contains the remaining instances.
6. The noise cases are discarded.

4.5 Predictive Filtering

This noise handling technique deletes misclassified instances from a data set after training the decision tree on a data set and testing if instances in a different data set are classified correctly. For this purpose the data set is split into n parts and as with the previous technique *n-1* parts are combined in order to create n overlapping parts of the original data set in order to enable cross-validation. Another requirement is, as mentioned earlier, the discretisation of the class variable, effort, to allow categorisation and therefore comparison of the actual values against the predicted values. Both steps can again be followed in the algorithm described Table 4.1. This filter works under the same principle as Brodley and Friedl's [23, 24, 25] partitioning filter. For the purpose of maintaining the simplicity and ease of comparison of the approach only a single filter algorithm identifies an instance as noisy.

Each of the overlapping parts of the data set (training sets) are used in turn to train a tree, which will be used to predict the class variable in the remaining parts of the data set (test sets). Instances in the test sets where actual values of the class variable are not equal to the predicted values of the class variable are considered as noisy and are filtered from the data set.

The algorithm shown in Table 4.4 describes this procedure.

Table 4.4: Algorithm for predictive filtering procedure

Algorithm for predictive filtering procedure

1. Build classification trees.
   - Build classification trees TS1_Tree to TS5_Tree for each training set TS1 to TS5.
2. Predict discretised effort.
   (a) Used TS1_Tree to predict discrete effort for DS5 resulting in probability values for each of the ten possible bins for each instance.
   (b) Used TS2_Tree to predict discrete effort for DS1 resulting in probability values for each of the ten possible bins for each instance.
   (c) Used TS3_Tree to predict discrete effort for DS2 resulting in probability values for each of the ten possible bins for each instance.
   (d) Used TS4_Tree to predict discrete effort for D3 resulting in probability values for each of the ten possible bins for each instance.
   (e) Used TS5_Tree to predict discrete effort for DS1 resulting in probability values for each of the ten possible bins for each instance.
3. Assign predicted bin for discretised effort by choosing the bin with the highest probability. If a tie occurs, all predicted bins are considered to be correct.
4. Compare predicted against actual for TS1, TS2, TS3, TS4 and TS5. Instances where predicted is not equal to actual are considered as noisy and are put into data set DSF_noise, the remaining instances are put into DSF_clean.
5. The noisy instances are discarded.

4.6 Filtering and Polish

The polishing technique is based on a list of noisy instances which will be altered, or polished, utilising a model built from a cleaned version of a data set. Whilst in Teng's original implementation only ordinal or nominal values were tested, this test was altered to accommodate the continuous nature of some of the variables in the software effort domain. As mentioned above this was done by discretising the dependent variable.

The data set is cleaned, with the predictive filtering technique (see Section 4.5). The clean data set is used to build a regression tree model which is used to predict the independent variables and the dependent variable in turn. The CART algorithm only uses a subset of all attributes to build the decision tree models. It establishes this subset by calculating each attributes influence on the dependent variable. As a result only attributes included in this subset are polished by using the predictions of the built regression tree. This polishing approach was introduced by Teng in [179] and successfully evaluated by Teng in [180, 181]. Whilst in the first two investigations (presented in Chapter 5) Teng's approach is taken as is and noise is handled by polishing all attributes including the dependent variable, in the final investigation (presented in Chapter 6) only the dependent variable is polished. This is done for two reasons. The first reason is the fact that the study presented in Chapter 6 is a simulation where random noise is artificial introduced exclusively to the dependent variable. The second reason is in relation to the notion that data quality is strongly linked to a domain and to the usage of a data. Since the problem domain in this thesis is effort prediction, noise in the dependent variable effort is the focus. Noise is here seen as values destroying the 'true' relationship between the independent and dependent variables. By eradicating the noise from this relationship from either side, the independent variables or the dependent variables this relationship is theoretically rebuilt.

Table 4.3 shows the steps for the filter and polish approach.

Table 4.5: Algorithm for filtering and polish procedure

Algorithm for filtering and polish procedure

1. Build regression tree DSF_clean_Tree using DSF_clean as training set. *The dataset cleaned by the predictive filtering technique was used to train a new tree.*

2. Predict values for attributes used by DSF_clean_Tree for DSF_noise one by one by swapping attributes from attribute to class variable. *Values for all attributes in turn are changed by predicting them using the tree built in the previous step. Each attribute is being used as dependent variable, allowing the polishing of all attributes.*

4.7 Summary

This chapter presented three automated data cleaning techniques for software engineering data sets. The three techniques are robust filtering, predictive filtering and filtering and polish each representing one of three different noise handling approaches. Robust filtering filters by combining training set and test set. Predictive filtering filters with different training set and test set, and filtering and polish alters the values of instances flagged as noisy.

Robust filtering builds a classification tree based which is pruned and instances in pruned nodes of the tree are eliminated. Therefore the pruned instances still have influence on a new model.

Predictive filtering eliminates instances by predicting values for the class variable. If predicted value does not match the actual value the instance is considered to be noisy, and it will be eliminated without influence on a new model.

Filtering and polish predicts new values for the eliminated instances from the predictive filtering technique, which minimises information loss.

The following two chapters (Chapter 5 and Chapter 6) describe three investigations in which the three data cleaning techniques were tested for their noise cleansing effectiveness. The Chapter 5 reports on two investigations which utilised a large real world data set, and Chapter 6 reports on an investigation which utilised simulated data.

Chapter 5

Studies Based on the Analysis of a Large Real World Software Engineering Data Set

5.1 Introduction

This chapter and the next present investigations carried out to test the noise handling abilities of the data cleaning techniques presented in the previous chapter. The investigations presented in this chapter are testing the data cleaning techniques on a new real world data set (EDS data set). A metrics expert familiar with the data set assisted with the investigations, and provided insight into the data as well as verification and clarification of assumptions made during the investigations presented in this thesis. The first investigation uses predictive accuracy as a measurement for the effectiveness of the noise handling techniques. Predictive accuracy has been used in machine learning to assess the effectiveness of cleaning algorithms [181, 60, 87]. It is a clear, tangible and easily understood measurement, but it might lack meaning, since it tends to distance the analyst from the real world problem domain and the actual issues of data collection. The second investigation presented in this chapter used a different proxy measure for the noise level. The EDS metrics expert provided benchmark values for plausible productivity values. These were then used to identify instances with implausible productivity values. The techniques' abilities to identify these instances will be compared. This is seen as a more meaningful measure to the software engineering domain.

The chapter starts with a description of the EDS data set which was used in both studies presented in this chapter. The EDS data set is new to the research community, and it was an excellent opportunity to investigate data quality issues in an unknown real world software engineering data set. Following the description of the data set are two sections on the two investigations, which used the EDS data set in which the methodology, the results and the conclusion of each investigation will be reported on.

5.2 The Data Set

This section provides some background information and a description of the software engineering data set used in the investigations presented in this chapter.

The data in the data set was provided by EDS in form of a Microsoft Access file extracted from a larger database (this file is from now on referred to as the 'database'). EDS is a large multinational computer service company, established in 1962, which is now known as HP Enterprise Services [71]. It has maintained a metrics program for several years. The database was provided by EDS in order to better analyse and understand their own data. The initial focus was software productivity and its determining factors, which influenced the extraction of related data. The database contains 213 tables which capture different aspects of software engineering projects.

Two main views of the database - contained data were seen as of interest for possible analysis and extraction. Project data was collected by EDS on a monthly basis providing a snapshot of each month, and data was also gathered at completion of each project. Since the focus of the initial analysis at the time of the data extraction was software productivity and its influencing factors, productivity data were of special interest. It was decided to extract data about closed projects only, in order to allow comparability. Tables 7.1, 7.2, 7.3, 7.4 and 7.5 in Appendix C list and describe the extracted attributes. Another two attributes, described in Table 7.6, were derived from the extracted attributes Unadj_FP_Count_Sum (count of unadjusted function points of a project) and SumOfEffort_Hours (count of effort hours spent on a project). These were derived in order to allow productivity analysis.

The attributes Project ID and Project Full Name are purely administrative and were extracted in order to aid identification purposes. These two attributes assisted the re-examination of particular projects in more detail. Throughout the work on this data set close contact with an EDS software metrics expert was maintained.

The extracted data set consists of 10434 entries for closed project from over 30 countries.

The extracted size measure for the single projects is unadjusted function points (UFP) [51]. UFP were chosen to allow a comparability of the size measure of the single projects.

Table 5.1: Descriptive Statistics for UFP

Statistic	**Value**
Mean	311.545
Standard Error	20.13
Median	128
Standard Deviation	797.82
Skewness	13.04
Range	19625
Minimum	1
Maximum	19626
Count	1571

EDS also captured size for some projects in lines of code (LOC), but this size measure was not extracted from the database since comparability of projects is dependable on the technology used and on adopted counting rules. EDS followed the IFPUG standard [74] to calculate the adjusted function point count. Since there have been discussions about the appropriateness of the function point adjustment it has been decided to extract UFP only. A more comprehensive discussion about the issues related to size measures can be found in [51].

The result of extracting only UFPs led to large numbers of missing values for the size attribute. Only 1571 (15%) of the 10434 projects therefore contain size values. There are also large amounts of missing values for effort. 8888 projects contained non-zero values for effort. The projects vary in size as can be seen in Table 5.1 which contains descriptive statistics for the UFP variable. The projects consequently vary also in the collected values for effort, which is shown in Table 5.2.

Table 5.2: Descriptive Statistics for Effort

Statistic	**Value**
Mean	5326.70
Standard Error	141.84
Median	1782.88
Standard Deviation	13372.18
Skewness	11.00
Range	410348.5
Minimum	0.5
Maximum	410349
Count	8888

The projects in the data set were collected from the beginning of the 1990's until 2004 with the number of start and closing dates peaking in 2003 as can be seen in Figures 5.1 and 5.2, which show the histograms for the data set's project start and close dates respectively.

The projects also vary in type and for which industry sector they were produced. The project types in the data set are as follows:

- Enhancement
- New Development
- Infrastructure
- Maintenance
- Production Support
- Other Projects & Services

Figure 5.3 shows a pie chart showing the proportion of projects according to type. It can be seen that most projects are related to production support and other services, thus explaining

Figure 5.1: Project Start Year Histogram

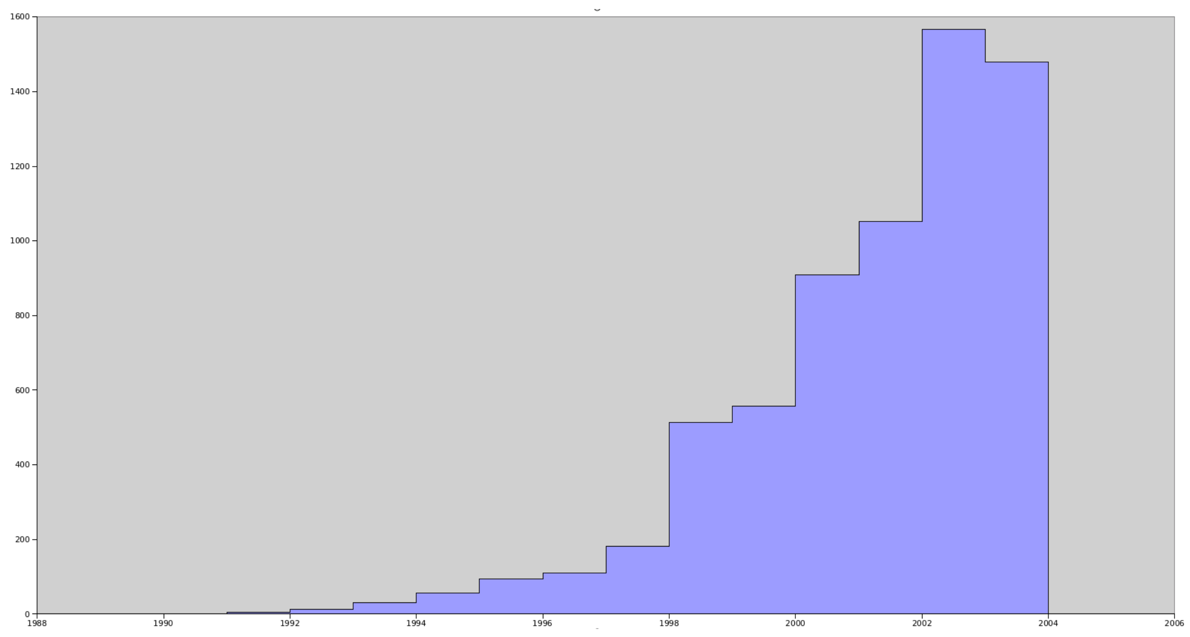


Figure 5.2: Project Close Year Histogram

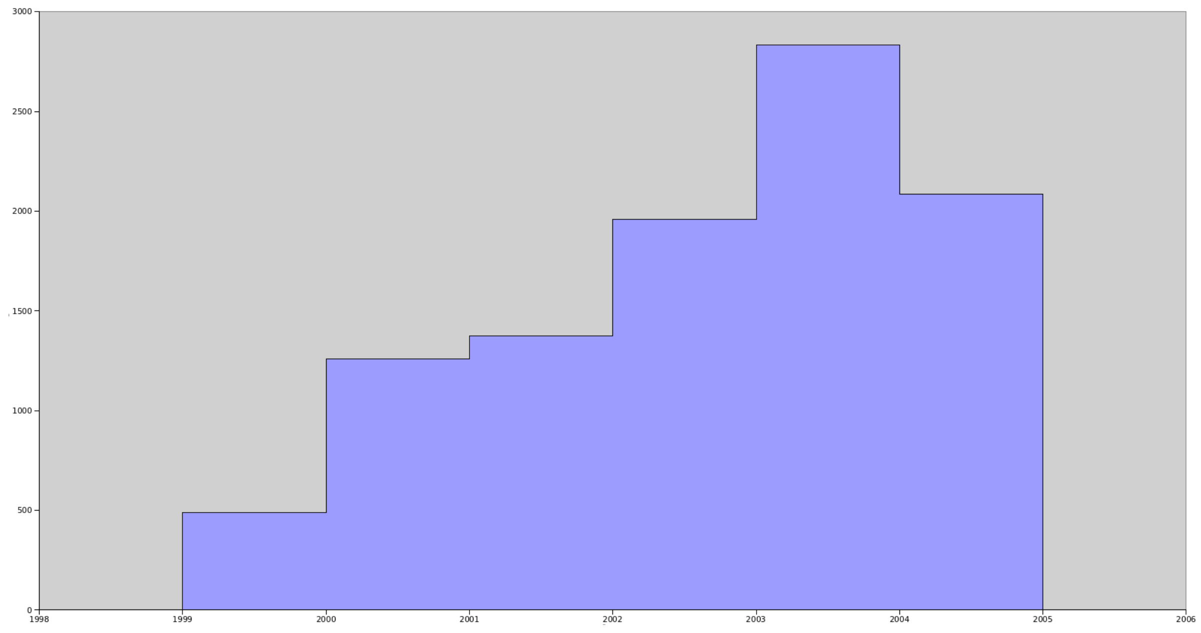
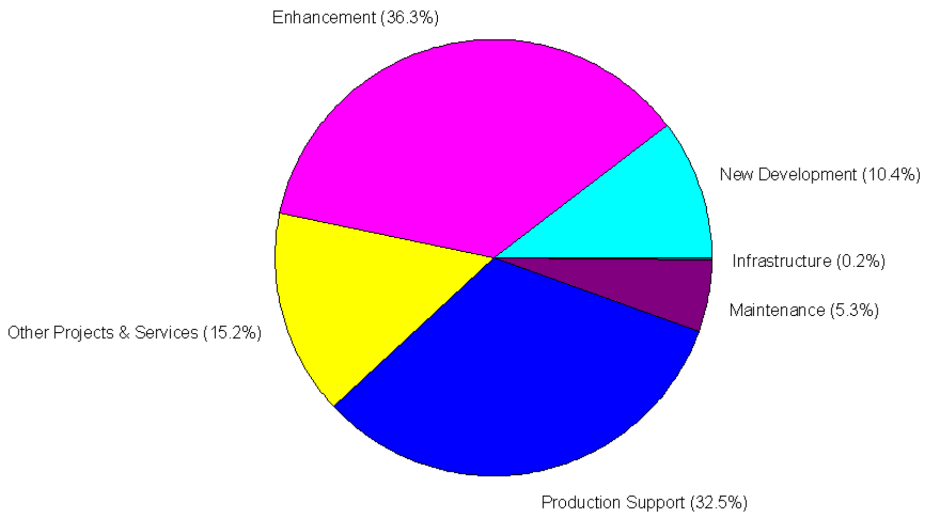


Figure 5.3: Pie Chart of Project Types



missing values for the size measure since they do not involve the development or fixing of source code.

Table 5.3 lists the industry sectors for projects in the data set, and how many projects for each industry sector exist in the data set. It can be seen that the manufacturing sector provides most projects (49%).

Table 5.3: Number of Projects for Industry Sector

Industry Sector	**Number of Projects**
Academic	6
Aerospace	152
Communications	962
Distribution	64
Military	29
Electrical	7
Engineering	115
Financial	1442
Government	540
Leisure	3
Manufacturing	5085
Health	440
Oil	20
Retail	109
Service	112
Transportation	676
Systems Integration	220
Utility	52
Other	314
Missing Entries	86 (1%)

5.2.1 Data Quality Issues in the Data Set

An early investigation of the data set [108] highlighted possible data quality issues. The analyses of the data set without addressing these data quality issues was seen as futile. This recognition was one of the motivations leading to the work presented in this thesis.

As already mentioned earlier in this chapter the data set contains large amounts of missing values. The most crucial here considered are effort and size. In one of the investigations presented this chapter (Section 5.4.2) missingness was considered as an contributing factor for the issues with one of the data cleansing techniques. In terms of this thesis missingness is recognised as a dimension of data quality and as a possible data quality issue, but it is beyond of the scope of the present work.

Other data quality issues recognised during the preliminary analysis presented in [108] were double entries and extreme productivity values, which were considered implausible by the metrics expert who assisted with the data analyses.

The issues in the data set can be caused by different reasons. A possible cause could be data entry issues where mistakes are made unintentionally due to unclear entry procedures and unclear entry units. For instance, effort could be recorded in hours or days. A confusion of the two would clearly lead to erroneous records. Another cause is the fact that collecting software project data is usually only one of many tasks of software engineers [102], and workload pressures paired with relative low priority of the data collection task can also lead to unintentional mistakes. A further possible cause for data quality issues can be deduced from information elicited during discussions with the software metrics expert. It is appears that possible management pressures led to software engineers obscuring exceptional performance data.

The following section will describe the first of two investigations presented in this chapter testing the merit of three different noise handling techniques .

5.3 Comparison of Predictive Accuracy of the Three Data Cleansing Techniques

This investigation is the first of two presented in this chapter. It was first reported on in [110]. It compared the predictive accuracy of models built after applying the three noise handling techniques predictive filtering, robust filtering and filter and polish. Predictive accuracy is a measure used to assess the effectiveness of classifiers and can be used to evaluate noise reduction techniques.

5.3.1 Methodology

The effectiveness of filtering, robust filtering, and filtering and polish were assessed using misclassification as a proxy measure of their effectiveness at dealing with noise. This study included 8911 instances rather then the 8888 instances used in the second study presented in this chapter. This is due to the inclusion of instances that contained zero values for effort[1]. These zero effort values were excluded from the subsequent study, since after discussion with the EDS metrics expert they were seen as missing values for effort. As mentioned in the previous chapter the discretisation of the class variable only produced five bins in this investigation. This was altered in the subsequent investigations to 10 bins since five bin discretisation was seen as too coarse for a continuous variable as effort.

The data were then artificially corrupted by manually introducing labelling or random errors into the attributes of 0 to 40% of instances. For nominal attributes, a noise level of x% means that the value of each attribute and the target class is assigned a random value x% of the time, with each alternative value being equally likely to be selected. For a numerical attribute, a random value was selected after discretisation of each attribute. Each bin resulting from the discretisation had an equal probability to be selected other than the the present value. This ensured that an attribute would be recognisable as noise even after discretisation, since an altered value could still fall in the same bucket as the not-noisy value.

With this scheme of inducing noise, the actual percentage of noise is always lower than the

[1]The reduction of the data set was not seen of much impact since 23 out of 8911 is less than half a percent.

theoretical noise level, as sometimes the random assignment would pick the original value (especially for nominal attributes). The actual percentages of noise ("Actual Noise Level (%)") are shown with the intended noise levels ("Artificial Noise Level (%)") in Table 5.4.

After the data set was subjected to the three different data cleaning techniques each cleaned version of the data set was split into test set (20%) and training set (80%) in order to produce five classifiers. The percentage of classification accuracy of all classifiers was averaged and noted to allow a comparison of the performance of the different noise handling methods under different noise levels.

5.3.2 Results and Conclusion

The results of this study are shown in Table 5.4. It presents the level of accuracy of the classifier built from cleaned versions of the EDS data set using predictive filtering, robust filtering, and filtering and polish. The accuracy level for the classifier built from the uncleaned data set is shown too to provide a comparison of the different cleaning methods against the do-nothing approach. It should be noted that higher accuracy implies less noise. The first row of the table reports the noise level used to corrupt the data and the second row shows the actual percentages of the corrupted training data (see previous section). It can be seen that filtering and polish, and robust filter performed equally well when no artificial noise was introduced. They showed higher accuracy to the do-nothing approach and the predictive filtering approach which was not significantly different to the do-nothing approach. As the noise level was increased it can be seen that the filtering and polish approach had higher classification accuracy in comparison to the other approaches. Summarising the results given in Table 5.4 it can be seen that filtering and polish out performed the other two data cleaning techniques. Filtering and polish was followed in overall performance by the robust filtering technique and then by the predictive filtering technique.

These results were considered as encouraging, but predictive accuracy only indicates how well a model built from a data set predicts instances of another data set and it does not consider domain specifics. The second investigation presented in this chapter was designed to assess the data quality from a software engineering point of view considering the specifics

Table 5.4: Classification accuracy - EDS data

Artificial Noise Level (%)	0	10	20	30	40
Actual Noise Level (%)	0.0	9.6	17.2	25.6	34.1
Methods:					
Do nothing	73.9	70.8	66.5	61.9	57.3
Robust filter	79.3	75.2	73.1	70.7	65.0
Filtering	74.8	71.6	69.2	66.7	62.5
Filter & Polish	80.3	77.4	76.1	73.1	67.9

of software effort data.

5.4 Comparison of Implausible Value Cleansing of Three Data Cleansing Techniques

This study was designed to approach the issue of validating the three noise handling techniques with a stronger focus on domain specific characteristics of the EDS data set. It was believed that the previous investigation provided a good indicator for the effectiveness of the noise handling techniques, but it did not accommodate any domain knowledge. This follow-up study was first reported on in [111]. It utilised the expertise of the EDS metrics expert mentioned above and his specific domain knowledge about the data set. Metrics experts have also been used by Khoshgoftaar and colleagues in [89], [191] and [192], but in these papers it is not clear what specific information was provided by the metrics expert. In [89] the domain expert identified noisy instances in a data set, but it is not apparent what characteristics constitute a noisy instance. In [191] and [192] the metrics expert evaluated ranks of potential noisy instances provided by the authors' noise handling techniques. How this evaluation was carried out and against what guidelines the ranks were tested is not clear. For investigation presented in the following sections the EDS metrics expert provided benchmark values for plausible values for productivity. Thus enabling the employment of these proxy measures to count the number of highlighted problematic instances and the noise handling techniques' ability to identify and cleans them.

5.4.1 Methodology

For the experiment presented in this section problematic instances in the data set were identified and flagged. Since the original data set contained historical data it was impossible to determine with complete certainty if an instance is noise. The EDS metrics expert provided minimum and maximum values for possible productivity levels to be used to compare instances remaining in the cleaned versions of the data set. Values that did lay outside these bounds were deemed to be implausible. These were then searched for in the remaining 8888 instances and in the cleaned versions of the data set produced by predictive filtering, robust filtering, and filter and polish.

5.4.2 Results and Conclusion

Table 5.5 shows a comparison of the three different noise handling techniques and the benchmark approach of doing nothing. Recall that since the EDS data set is drawn from the real world any definitive statements cannot be made concerning the 'true' noise level. Consequently proxy measures were used based upon the ability of a rule tree classifier to correctly classify the project effort of each instance and also to isolate implausible instances. The initial data set contained $n = 8888$ instances after projects are eliminated where development effort was unknown. The number of instances eliminated, e depended upon the technique. However, note that the filtering and polish method eliminated zero instances (also the do-nothing strategy) since values were edited rather than removed. This might be seen as an advantage compared with the robust filtering method which eliminated more than 6200 instances (i.e. in excess of 70% of all cases) and the filtering method which eliminated more than 5800 instances (i.e. in excess of 65% of all instances)[2]. Next it can be observed the relative number of implausible instances, i that were not identified and therefore not eliminated. For the do-nothing technique all 347 remained whereas the robust filter was able to eliminate just over 88% of such instances. This is indicative of the effectiveness of the approach inasmuch as it may be believed that this gives an indication of the ability of the technique to remove the non-implausible noisy and therefore unidentified instances. The surprising value for i is for the filtering and polish technique which actually generated new (i.e. not previously contained in the data set) implausible instances. This is a consequence of the way in which new values are imputed for those cases that are filtered. It is also consistent with Teng's results [179, 180]. Ultimately it is not believed that this fact is too serious since implausible instances can always be detected algorithmically. The other instances that are contained in the data set that are problematic were those that had zero productivity values. These may be considered as a special instances of implausible value, however, the cause is due to missing values rather than noise. As stated previously the data set contained a substantial number, 7436, of problematic instances where size information was unavailable

[2]This very high level of instance elimination may seem surprising but it should be noted that the majority of instances did not even contain size information so that even productivity rates could not be calculated.

so productivity cannot be computed. Since these cases were demonstrably of low quality one might expect an effective noise detection approach to eliminate a significant proportion of these instances. The number remaining is given as z in Table 5.5 such that low values are to be preferred. It can be seen that the identification of these values by all three methods was similarly effective. The filter and polish approach left the same number of zero productivity values in the cleaned data set as the filter approach. This is due to the technique applied to polish the noisy instances. Whilst the values of noisy instances are altered, the instance which were identified by the filter method as not noisy are left as is. It has to be noted that even if all three methods identified nearly equal numbers of zero productivity levels the ratio of zero levels against remaining instances was best for the filter and polish method. This is also partially reflected in the overall level of problematic instances after the application of the three methods. Whilst the filter methods' performance was just under the baseline, the robust filter method resulted in a increase of the ration of problematic instances against remaining instances. The best performance could be observed for the filter and polish method. Therefore it can be argued that when using either misclassification rate or proportion of implausible instances not eliminated the filter and polish technique is to be preferred. Another interesting observation of this investigation is that variables which might be considered as crucial for effort prediction were not always used in the building of the trees. The size measure was in some cases omitted from the final models. This might be due to the large amount of missing and therefore imputed data.

Table 5.5: Misclassified v. Classified Instances by Data Quality Technique

	Noise Handling Technique			
Measurements	'do-nothing'	Predictive Filtering	Robust Filtering	Filtering and Polish
Total # of instances *(n)*	8888	8888	8888	8888
Noisy instances eliminated *(e)*	0	5873	6243	0
# of instances remaining *(n-e)*	8888	3015	2645	8888
# of implausible instances remaining *(i)*	347	113	39	1267
# of zero-productivity instances remaining *(z)*	7436	2469	2430	2469
% of implausible instances remaining *(i / n-e) x 100*	3.90	3.75	1.47	14.26
% of implausible instances remaining of original identified *(i/347) x 100*	100	32.56	11.24	365.13
% of zero-productivity remaining of original identified *(z / 7436) x 100*	100	33.20	32.68	33.20
% of implausible instances and zero-productivity instances remaining of original identified *((i+z) / n-e) x 100*	87.57	85.64	93.35	42.03

5.5 Conclusion of Investigations Based on a Real World Data Set

This chapter presented two investigations which were based on a new large software engineering data set and which tested the data cleansing capabilities of the three noise handling techniques presented in Chapter 4 against the do-nothing approach. The two investigations had to utilise proxy measures since the true noise level of a real world data set cannot be known.

The first investigation used predictive accuracy as a measure for the data cleansing capabilities. The data set was induced with additional artificial noise, and after application of the techniques the classifier's predictive accuracy was tested. The results indicated that the filtering and polish technique performed best in increasing predictive accuracy. The robust filtering technique performed second best followed by the predictive filtering approach. All three data cleansing techniques resulted in higher predictive accuracy than the do-nothing approach.

For the second investigation, implausible values were identified by a metrics expert familiar with data set. After applying the noise cleansing techniques the numbers of identified implausible values were compared. The robust filtering technique performed best in lowering the number of implausible values, followed in performance by the predictive filtering technique and the do-nothing approach. The filtering and polish technique actually increased the number of implausible values. However this negative impact was moderated by the fact that implausible values are easily identified and since the filtering and polish technique retained all instances whilst eliminating zero values in the dependent variable. Nevertheless, the inconsistencies in the results between the first and the second investigation are puzzling.

Since the true noise level of a real world data set cannot be known with 100% certainty, there are limitations to accuracy of investigations based on real world data. Real world data provide opportunity to test techniques in a real context, but introduce uncertainty to the measurements. Therefore a subsequent investigation should deal with this uncertainty.

The next chapter presents an investigation which introduced certainty by simulating data sets with known noise levels. The creation of the simulated data sets were based on the data set presented in this chapter in order to assure continuity of the overall research questions, and to ensure domain specific characteristics are maintained.

Chapter 6

Evaluating Noise Handling Techniques With Simulated Data

6.1 Introduction

The previous chapter reported on investigations which tested the data cleaning capabilities of robust filtering, predictive filtering and filtering and polish on a large real world data set. Whilst the investigations indicated that the techniques can improve the quality of a given data set against the do-nothing approach, the fact that the true extent of noise in a given real world data set cannot be known is a limitation of using real data for the evaluation of noise handling techniques. Another issue with real world data is the influence of factors which might obscure the true noise detection capabilities such as missingness, which is beyond the scope of this thesis. A solution to these issues of uncertainty of noise in a real world data set is to use simulated data to evaluate the noise handling techniques.

This chapter reports on an investigation which used simulated software engineering data sets, which were modelled on the real world software engineering data set presented in Chapter 5 as basis for tests of the performance of the three noise handling techniques. Simulation of the data sets ensured that the true values for each instance in a data set is known. The simulated data sets were then introduced with artificial noise, with varying noise levels, such that the true noise levels are known to the analysts.

The simulation process provides control in empirical investigations, and it has been successfully used in the software cost domain by Pickard et al. [143] and Shepperd and Kadoda [163]. It can also be used to control different aspects of noisy environments, like missingness and unknown noise levels, which can confound the results of investigations. The investigation presented in this chapter only tests the data cleaning techniques on random noise. Systematic noise is beyond the scope of this investigation.

The chapter starts by describing the methodology, which includes a description of the data simulation process, the noise imputation method and the noise cleansing process. This is followed by a results and conclusion section.

6.2 Methodology

The investigation presented in this chapter relies on a blind evaluation, where the data cleaning assessments were carried out without knowledge about the production of the test data. This blind evaluation was achieved by separating the simulation process from the data cleaning process through allocation of these tasks to two different researchers. The simulated data sets were created and induced with artificial noise by one researcher, and another researcher applied the data cleaning techniques without knowledge of the underlying model of the data and without knowledge of the induced noise levels. The test data was comprised of four simulated data sets which were induced with artificial noise. The separation of data creation and noise handling technique assessment is believed to be a strength of this investigation since it aided an unbiased evaluation.

6.2.1 Simulation of Test Data Sets and Artificial Noise Imputation

The simulated data sets were modelled on a cleaned version of the EDS data set, which was presented in Chapter 5. The EDS data set was cleaned using the predictive filtering technique, since this technique did not produce new noisy instances like the filtering and polish technique, and since it proved to be less information expensive, meaning it eliminated fewer instances, than the robust filtering technique as shown in the investigations presented in the previous chapter[1]. Additionally the EDS data set was also cleared of instances containing missing values in any of its attributes. This was done to avoid the influence of missingness on the results, which was beyond the scope of this investigation. The resulting data set contained 123 instances[2].

The next step of the preparation of the simulated data sets was the examination of the relationship of Data 123's attributes. From previous investigations it was found that the size attribute was the single most influential attribute in the prediction of the response variable effort. The previous investigations and the work by Kitchenham [97] led to the assumption that the underlying relationship between effort and size is a linear one. Therefore the four

[1]All three techniques were described in Chapter 4

[2]The cleaned version of the data set will from now on be called Data 123, since it contains 123 instances.

simulated data sets were created using the model $Y = a_1X_1 + a_2z_1 + a_3z_2$ where Y is effort, the response variable, X is size and z_1 and z_2 are two levels, in form of dummy variables taking on the values of either '1' or '0' of a categorical variable Z (note that the model can contain more than one categorical variable, which would be added in the same way as Z by creating dummy variables.). Each simulated data set had a different linear distribution. The simulated data sets were all of the same size, 123 instances, and had no missing values. It is important to note again that, whilst missingness is an important issue, it is beyond the scope of the investigations of this thesis. As indicated in the previous chapter it is suspected that missingness contributed to issues with one of the three noise handling techniques when the filtering and polish technique created more implausible values then were originally present in the data set.

Noise Imputation

The four simulated data sets were induced with noise in the response variable at levels of 10, 30, 60 and 90 percent. Considering that noisy values are essentially values destroying and confounding the relationship between the input attributes and the response variable adding artificial noise to the response variable is sufficient, since the model used to create the relationship is not appropriate any more. Also the artificial noise introduced into the simulated data sets consisted only of random noise since systematic noise is beyond the scope of the presented investigation.

For the simulated data sets noise was produced using a random number generator creating normally distributed values between +3000 and -3000. Table 6.1 shows that the actual values generated by the random number generator were between 2679.06 (highest maximum) and -1488.65 (lowest maximum). The statistics in this table indicate normally distributed values were produced. Since this thesis did not investigate the causes of noise, only random noise has been investigated. Systematic noise, since it follows some system and therefore is influenced by some mechanism, is beyond the scope of this investigation. The suspicion is, that systematic noise is more complex to deal with than random noise, and more knowledge about the underlying mechanisms need to be available in order to identify systematic noise.

For simulated data set one (SDS1) 12 instances were induced with noise producing an approximate noise level of ten percent (Table 6.1 also shows the actually achieved noise level). Simulated data set two (SDS2) had 37 instances induced with noise resulting in an approximate noise level of 30 percent. Simulated data set three (SDS3) had 74 instances induced with noise resulting in an approximate noise level of 60 percent. Simulated data set four (SDS4) had 111 instances induced with noise resulting in an approximate noise level of 90 percent. The instances to be corrupted with noise were chosen at random. If the addition of noise in an instance resulted in negative or zero values in the response variable due to negative noise values a different instance was chosen for the addition of noise, since negative or zero effort is not possible in the real world.

6.2.2 Data Cleansing

As mentioned above this part of the investigation has been carried out without the author's knowledge of the underlying distribution of the data in the clean data sets. This was done in order to avoid bias, and to avoid unconscious assistance for one or all cleansing methods. The three techniques robust filtering, predictive filtering and filtering and polish were applied as described in Chapter 4.

Table 6.1: Summary of the Simulated Data Sets

Statistic	SDS1	SDS2	SDS3	SDS4
Mean	-52.84	182.01	131.43	-120.78
Standard Error	304.39	160.52	115.31	87.67
Median	-67.41	220.06	134.46	-101.11
Standard Deviation	1054.44	976.42	991.92	923.68
Sample Variance	1111845.24	953386.43	983908.92	853176.17
Kurtosis	-0.93	0.10	-0.57	0.19
Skewness	0.33	-0.26	0.06	0.15
Range	3402.28	4476.02	4839.91	4758.05
Minimum	-1488.65	-2076.57	-2160.85	-2318.90
Maximum	1913.63	2399.45	2679.06	2439.15
Sum	-634.12	6734.25	9726.03	-13406.07
Count	12	37	74	111
Total number of instances in the data sets	123	123	123	123
Noise in X % of instances	10	30	60	90
Actual Noise Level in %	9.76	30.08	60.16	90.24

6.3 Results

This section discusses the results of the investigation presented in this chapter. First a comparison of the noise detection capabilities of two of the data cleaning techniques are presented, followed by a presentation of the noise reduction

6.3.1 Noise Detection Capabilities

This section of the paper compares the noise detection capabilities of two of the three data cleaning techniques. Only robust filtering and predictive filtering were compared for their noise detection capability since filtering and polish does not discover noisy instances, but only alters the instances flagged as noisy by the filtering technique. It was investigated if the two techniques could correctly identify noisy instances. Robust filtering and predictive filtering eliminate instances which they identify as noisy, thus categorising an instance as noisy or not noisy. This was compared to the actual noisiness, i.e. noisy or not noisy, allowing a comparison of matches and mismatches.

Tables 7.7 to 7.18 show the match/mismatch of the instances highlighted by robust filtering and predictive filtering and the instances actually containing noise. Tables 7.7, 7.10, 7.13 and 7.16 in Appendix D show the match/mismatch for the robust filtering technique. The tables 7.8 to 7.18 show the match/mismatch for the predictive filtering technique. The predictive filtering technique required the descretisation of the response variable effort into deciles. The predictive filtering technique then predicted the values and any mismatch was highlighted as noisy. Whilst Tables 7.8, 7.11, 7.14 and 7.17 in Appendix D show the agreement for all misclassified instances, Tables 7.9, 7.12, 7.15 and 7.18 in Appendix D only show the instances where the misclassification was greater or equal than two[3].

Table 6.2 summarises these results in the statistics precision, recall, F1-score and Cohen's Kappa. Whilst precision, recall and F1-score allow the comparison of the detection capa-

[3]Due to n-fold cross validation n classifiers are built. Each classifier either predicts correctly or not. If correctly predicted an instance is classified as noisy, otherwise the instance classified as not-noisy. Since each classifier predicts instances for *n-1* parts a a data set classification can vary between *0* and *n-1*, where *0* is the result of no classifier flagging the instance as noisy, and where *n-1* all classifiers flag the instance as noisy.

bilities of the two techniques, they do not provide a general judgement of the techniques. The Cohen's Kappa statistic with Landis and Koch's [101] interpretation[4] of the Cohen's Kappa statistic provide a benchmark to compare the classification agreement against. The statistics for the robust filtering (RF in Table 6.2) technique for SDS1 could not be calculated since zero noisy instances were correctly identified. This comparably bad performance of the robust filtering technique can also be observed in the results of the other data sets. Whilst its performance increased with increasing noise level[5] it was still performing worse than the predictive filter technique. It can be seen that the Cohen's Kappa for the robust filtering technique never exceeds 0.06. Therefore the agreement is negligible. The predictive filter approach rejecting all misclassifications (PF +1) and the predictive filtering approach rejecting only misclassifications greater or equal than two (PF +2) performed better than the robust filtering technique. Whilst precision is higher in all data sets for PF +2 recall is higher in all data sets for PF +1. Apart from the 10% noise level in SDS1 where F1-score and Cohen's Kappa is lower for PF +1 than PF +2 and the lower Cohen's Kappa for the 30% noise level in SDS2, PF +1 slightly outperforms PF +2. It has to be stated that both approaches do not exceed fair agreement either. Overall, it appears that the PF +1 approach was the best performing approach.

[4]The Kappa value benchmarks for agreements vary depending on the number of categories and their weighting, and they might be based on researchers experiences [167]. Often quoted are the Landis and Koch standards, which provide the following interpretation of kappa values: < 0 = poor, 0.00 to 0.20 = slight, 0.21 to 0.40 = fair, 0.41 to 0.60 = moderate, 0.61 to 0.80 = substantial, and 0.81 to 1 = almost perfect [101].

[5]Remember; noise levels for data sets: SDS1 10%, SDS2 30%, SDS3 60% and SDS4 90%

Data set/Technique	Precision	Recall	F1-Score	Cohen's Kappa	Landis-Koch Interpretation
SDS1 RF	n/a	n/a	n/a	∼-0.07	no agreement
SDS1 PF 1+	∼0.17	∼0.83	∼0.14	∼0.14	slight agreement
SDS1 PF 2+	∼0.38	∼0.42	∼0.20	∼0.33	fair agreement
SDS2 RF	∼0.38	∼0.14	0.10	∼0.05	slight agreement
SDS2 PF 1+	0.50	∼0.58	∼0.27	∼0.31	fair agreement
SDS2 PF 2+	1	∼0.27	∼0.21	∼0.34	fair agreement
SDS3 RF	∼0.68	∼0.26	∼0.19	∼0.06	slight agreement
SDS3 PF 1+	∼0.76	∼0.59	∼0.33	∼0.29	fair agreement
SDS3 PF 2+	∼0.96	∼0.30	∼0.22	∼0.24	fair agreement
SDS4 RF	∼0.90	∼0.32	∼0.23	∼0	no agreement
SDS4 PF 1+	∼0.95	∼0.66	∼0.39	∼0.14	slight agreement
SDS4 PF 2+	1	∼0.37	∼0.27	∼0.10	slight agreement

Table 6.2: Comparison of Precision, Recall, F1-Score and Cohen's Kappa

6.3.2 Noise Reduction Capabilities

This section of the chapter compares the noise reduction capabilities of the three noise handling techniques. For this purpose the residuals noise levels of each technique in data sets SDS1, SDS2, SDS3 and SDS4 were compared against the do-nothing approach, where the data set was left as is. Tables 7.19, 7.20, 7.21, 7.22, 7.23, 7.24, 7.25, 7.26, 7.27, 7.28, 7.29 and 7.30 in Appendix D show the results of a Mann-Whitney tests to show the mean comparisons of the residual noise levels against the do-nothing approach.

Table 6.3 summarises the results of the outcomes of the Mann-Whitney tests. It is shown if the noise level was increased or decreased. A statistically significant difference of the means within a 95% confidence interval is indicated by a “Y”. A “N” indicates that the difference was not significant. For each data set an interpretation of the performance is given as either “Best”, “Middle” or “Worst”. The predictive filtering technique appears to be the most effective technique in reducing the noise levels of the data sets. The reduction is not always statistically significant, but the technique was consistent in its outcome. The robust filtering technique was less consistent in its outcome, decreasing the noise level for data sets SDS2 and SDS3, but increasing the noise levels for data sets SDS1 and SDS4 [6]. The filtering and polish technique was consistently increasing the noise levels. The differences between the noise levels of the untreated data set (do-nothing) and noise levels of the the data set cleaned with the filtering and polish were not statistically significant, but the increase of noise is the opposite of what was actually attempted and is seen as very problematic.

[6]As mentioned before, the noise levels were 10% for SDS1, 30% for SDS2, 60% for SDS3 and 90% for SDS4.

	Robust Filtering		Predictive Filtering		Filtering and Polishing	
Data set	Significant Difference	Technique Comparison	Significant Difference	Technique Comparison	Significant Difference	Technique Comparison
SDS1	Y Increased Noise	Worst	Y Decreased Noise	Best	N Increased Noise	Middle
SDS2	Y Decreased Noise	Best	Y Decreased Noise	Best	N Increased Noise	Worst
SDS3	Y Decreased Noise	Best	N Decreased Noise	Middle	N Increased Noise	Worst
SDS4	Y Increased Noise	Worst	N Decreased Noise	Best	N Increased Noise	Middle

Table 6.3: Summary of the Residual Mean Comparisons

6.4 Conclusion of Investigations Based on Simulated Data Sets

This chapter reported on an investigation based on simulated data sets which were corrupted with artificial random noise testing the three data cleaning techniques presented in Chapter 4. The data sets were based on the real world software engineering data set which was used in the previous two investigations reported on in Chapter 5. Simulated data was used since it allowed to control confounding factors. It also provided a test environment with known noise levels.

The first part part of the investigation compared the noise detection capabilities of robust filtering and predictive filtering. The results showed that predictive filtering performed better than the robust filtering technique in identifying noisy instances, but none of the two techniques performed convincingly well. The second part of the investigation compared the noise reduction capabilities of robust filtering, predictive filtering and filtering and polish. Again the predictive filtering technique performed best, followed by the robust filtering technique leaving the filtering and polish technique performing worst, but the changes in noise levels were not always significant.

Overall concluding from the results of the simulated data investigation it can be said that none of techniques performed convincingly well. In fact, concluding from these results robust filtering and filtering and polish performed very poor, and based on the presented results, they would not be recommended to be used. The predictive filtering technique was the best performing technique out of the three, but its performance was also not convincing.

Comparing these results to the results of the investigations presented in Chapter 5 these results appear confusing. Whilst in the first investigation presented in Chapter 5 the filtering and polish technique clearly performed best in improving predictive accuracy, its ability in dealing with implausible values was very weak. In fact, it created more implausible values. One redeeming feature of the filtering and polish technique was that it retained all instances of the data set and decreased the number of zero values. The poor performance of the filtering and polish technique was repeated in the investigation which was based on simulated data. This reversal of performance could also be observed comparing the results of predictive filtering in all three investigations. Whilst it performed least well in improving predictive accuracy, it performed second best in reducing implausible values, and it was the best performing technique for the simulated data investigation. The robust filtering technique was second best in improving predictive accuracy, it was the best technique in identifying implausible values, and the performance of the robust filtering technique for the simulated data investigation were a weak second best.

Very concerning are the differences of the outcomes of the investigation using predictive accuracy and the other two investigations, raising doubts about the usefulness of measures like predictive accuracy in measuring the effectiveness of data cleaning techniques. It also provides an indication of the magnitude of the issues data quality and noise. The underlying model in the simulation was very simplistic (i.e. linear), and the noise was symmetric and random. This should have helped the noise detection techniques, but they still performed badly.

The next chapter will summarise the findings of this thesis, discuss raised issues and conclude with comments about the wider impact of the presented work. It will also discuss limitations of the presented investigations with possible pointers to future research into data quality in software engineering data sets.

Chapter 7

Conclusions

7.1 Summary of Research

The work presented in this thesis investigated data cleaning in empirical software engineering with a specific focus on noise cleaning. It has been established that data quality can be defined as 'fitness for purpose' of the data. Therefore domain purpose and data characteristics are important factors for establishing the quality of data. In this thesis three noise cleaning techniques namely predictive filtering, robust filtering and filtering and polish have been evaluated for their ability in dealing with noise and not just outliers in software effort data. Each of these techniques represent an approach for dealing with noise. Firstly, noise can be identified by building a model which is built on a data set and instances in this data set not following the rules of the majority of instances are filtered or ignored. Noise contributes to the development of an initial model which will be penalised for being over complex. This approach is represented by robust filtering. Secondly, noise can be filtered by training a model on a different set of data then another set of data to be tested, such attempting to minimise the association between underlying model and identified noisy instance. This approach is represented by predictive filtering. For the third approach, noise is altered, polished, after being identified first. This is done in order to retain maximum of information. This approach is represented by the filtering and polish approach.

Since quality is domain dependent, it was essential to establish which instances were considered poor data quality, or in other words which instances were noisy. Noise can be defined as an 'unwanted' disturbance in data. Outliers can also work as disturbance for models, but they are not necessarily unwanted in software engineering data since they can for instance indicate the start of new trends. In this thesis the position is taken that outliers and noise differ in their creation and meaning for data analyses. Yet, they are often treated as one by many researchers especially those who come from a machine learning background. Outliers are instances which exhibit exceptional values from the rest of a given population, and such might appear suspicious, but might be the result of valid data creation mechanisms which are either not understood or neglected. Noise are data which can appear exceptional, but result from intentional or unintentional erroneous data creation or corruption.

Since there has been little research carried out in the effectiveness of automated noise handling techniques in the empirical software engineering community, this thesis presented three separate investigations assessing the data cleansing capabilities of the three noise handling techniques, robust filtering, predictive filtering and filtering and polish, which were presented in Chapter 4. These techniques represent each a different noise handling approach and their effectiveness in cleaning software effort data from noise was tested.

A comprehensive literature review has been carried out to survey the empirical software engineering literature in order to establish how noise, in the above described sense, has been dealt with in the empirical software engineering community. The results of this systematic literature review were presented in Chapter 3. The literature review clearly showed that there is a gap in the research in empirical software engineering concerning the treatment of noise. Whilst out of the 161 retrieved papers the majority considered poor data quality as an issue, very little has been done to combat this issue. In particular the field of automated noise detection and handling and the empirical assessment of poor data quality was under-represented. Only 15 papers employed automated data cleansing techniques and only 21 out of 161 papers attempted to assess data quality of software engineering data sets.

The first two investigations presented in Chapter 5 utilised a large industrial software engineering data set (EDS data set), provided by a large multinational computer service company, containing software project effort data. Following initial analysis of the EDS data set it was found to contain substantial levels of noise. Since the true noise level of a given real data set cannot be known [37, 66] proxy measures had to be utilised to test the noise cleansing capabilities in the two investigations utilising the EDS data set.

The first investigation using the EDS data set utilised the predictive accuracy of a classifier built from data sets cleaned with the three techniques. The most successful technique was the filtering and polish technique, followed by robust filtering and predictive filtering. All three techniques improved predictive accuracy against the do-nothing approach.

The second investigation, using the EDS data set, used the number of identified problematic instances as proxy measure for the data cleansing performance of the noise cleaning techniques. A metrics expert familiar with the EDS data set provided thresholds values for implausible productivity values enabling the identification of implausible instances. The techniques' ability to identify the instances with implausible productivity values and zero productivity values was then compared. The robust filtering technique was the most effective technique in identifying implausible values, followed in performance by the predictive filtering technique. The filtering and polish technique actually increased the number of implausible values, but retained all instances of the data set. Robust filtering and predictive filtering reduced the number of instances significantly hence being less information efficient.

The third investigation was based on simulated data sets, which were based on a cleaned version of the EDS data set. Simulated data has the benefit, that the 'true' noise level of a data set can be known, and specifics of a given domain can be incorporated. Since the simulated data sets were based on the EDS data set, the domain specifics of the software effort data were retained. The uncertainty about the 'true' noise level in real world data complicates the assessment of noise cleaning techniques since proxy measures need to be utilised which approximate noise cleaning capabilities. In simulated data noise levels are known. They therefore provide analysts with a precise comparison of noise cleaning techniques. This investigation was split into two parts. First, the noise detection capabilities of robust filtering and predictive filtering were compared. Since filtering and polish utilises the predictive filtering technique for identification of noisy instances, its noise detection capabilities did not need to be compared. The predictive filtering technique performed marginally better than the robust filtering technique, but both techniques did not preform convincingly well in identifying noisy instances. The second part of the investigation compared the noise cleaning capabilities, that is the reduction of the overall noise levels, of all three techniques. Again the predictive filtering technique performed better then robust filtering and filtering and polish, and again none of the techniques performed significantly well.

A comparison of the results of the three investigations show that none of the techniques performed particularly well. The first investigation utilising predictive accuracy resulted in filtering and polish outperforming the other two techniques. In the two following investigations filtering and polish performed least well, but it had one redeeming feature, it retained all instances. How helpful this feature is when the overall noise level is increased appears questionable. Whilst performing better than the do-nothing approach in both investigations based on the EDS data set, the two filtering techniques also did not perform exceptionally well in the simulated data investigation.

In conclusion, what can be learned from these results? When applying automated noise cleaning techniques caution about the confidence in these techniques should be taken. It has been shown in this thesis that noise cleaning techniques can improve predictive accuracy, but this does not necessarily mean that noise levels are lowered. Relying purely on predictive accuracy can lead to a false sense of success as highlighted in differences of the results of the three investigations. Whilst the first investigation clearly showed the merit of all presented noise handling techniques the subsequent investigations reduced this confidence into the findings of the first investigation. It should be remembered that predictive accuracy is purely a measure of how well a classifier predicts new unseen instances. A combination of investigations based on real world software engineering data and simulated data should be utilised in order to investigate the 'true' effectiveness of noise cleaning techniques.

This was recently done by Yoon and Bae [205][1] when they compared the outlier detection capabilities of five techniques against their own outlier detection technique in the software effort domain. One of the five techniques was the filtering and polish technique, but essentially Yoon and Bae compared the outlier detection capability of the predictive filtering

[1]This paper was published after the bulk of the work for this thesis was completed already.

technique, since it is used by the filtering and polish technique. The noise correction abilities of the filtering and polish technique were not tested. The other techniques compared were hierarchical clustering, a frequent pattern analyser (less frequent patterns are seen as noise) and the two techniques proposed by Khoshgoftaar and colleagues, PANDA and AOI[2]. Yoon and Bae's technique combines a measure of normality for each attribute with a pattern analysis. When Yoon and Bae refer to outliers they seem to mean noise in the sense used in this thesis since they introduced artificial noise into three real world project data data sets and 48 simulated project data data sets. They criticise the filtering and polish technique for comparing the interaction of attributes with the class variable one by one, when in fact interactions between attributes in software project data exists, but in this thesis a more practical approach was taken, where the the noise detection capabilities of the techniques were tested without focussing on theoretical soundness. This practicality in approach is not new in empirical software engineering. For instance ordinary least square regression, which is based on the assumption that variables are orthogonal to each other when this is not necessarily so, has been utilised widely in software effort predictions. The results from the real world data showed that Yoon and Bae's approach were the most effective and accurate in identifying noise in all attributes. The predictive filtering technique detected most noisy instances in dependent variable, but was least effective in identifying noise in *all* attributes. This is related to the issue of class noise and attribute noise as discussed in Chapter 2. Since independent and dependent variables can be swapped this is not seen as problematic. Therefore it is believed that the full potential of the predictive filtering algorithm was not observed. The AOI technique (which utilised PANDA) was least accurate since it identified too many instances. Yoon and Bae's simulated data evaluation indicated that with increasing size of the data set their approach, AOI and hierarchical clustering showed increased detection accuracy. PANDA and the frequent pattern analyser did not change in detection accuracy, and predictive filtering actually decreased detection accuracy. The decrease of detection accuracy is due to the fact that predictive filtering identified the same number of instances as noisy in different sizes of data sets, indicating that increased number of instances in the

[2]Chapter 3 Briefly describes the papers where these two techniques were proposed.

training set does not aid the discovery of new noisy instances. With increased noise levels Yoon and Bae's technique's detection accuracy stayed stable. Predictive filtering was the most effective in identifying noise in the dependent variable, but least effective in identifying noise in dependent variables.

Yoon and Bae's efforts are remarkable, especially since they systematically compared noise cleaning techniques in different environments and combining the benefits of real world data and simulated data. The problem in relation to the results of the filtering and polish technique is that the purpose of the method has been misinterpreted. The technique has been evaluated against its noise detection capabilities, but as mentioned above, filtering and polish utilises predictive filtering for the detection of noisy instances, and its actual purpose is to alter the identified instances. Also, it was criticised for only identifying noise in the dependent and not in the independent variables, but again as mentioned above, this can be changed by swapping dependent and independent variables. The relative poor performance the technique showed in Yoon and Bae's investigation is hardly relevant, as long as techniques are compared for what they are.

Techniques can be improved. The aim of this thesis was to investigate data quality and data cleaning in software engineering data. Therefore three noise cleaning techniques were chosen which represented three noise cleaning approaches and their 'true' noise handling capabilities were investigated. For instance filtering and polish could possibly be improved by utilising more effective noise detection techniques, and more effective missing value imputation techniques could be applied to correct filtered instances.

Apart from the issue of evaluation of noise cleaning techniques, a practical conclusion of the work presented in this thesis for empirical software engineering should be the incorporation of automated noise cleaning techniques into a holistic approach such as Kimball [96], with elementising, standardising, verifying, matching, householding and documenting as steps of preprocessing of data sets before analyses. Whilst the first four steps could be partially fulfilled by automated noise cleaning techniques, householding and documentation have to follow in order to make investigations replicable and thus increasing academic rigour. The evaluations of the noise cleaning techniques presented in this thesis showed that that noise cleaning techniques should be applied cautiously, since effectiveness of a noise cleaning technique indicated in an investigation does not necessarily prove its 'true' noise cleaning effectiveness. Householding therefore would incorporate inspection of instances flagged as noisy, and assuring noise correction was adequate if applied. Documentation also has to happen such that assumed noisy instances can be re-inspected in future analyses, such limiting information loss if misclassified instances are re-validated.

7.2 Contributions

This section lists the contributions of this thesis.

- **Provided clarification and distinctions of the terms noise and outliers**
  In this thesis noise and outliers are defined and their differences are clarified. Noise are instances in a data set which result from intentional and non-intentional entry errors. They are problematic since they distort and falsify the true underlying relationships of attributes in the data set. Outliers are instances which are exceptional and could be considered as problematic since they can have a leverage effect on the analysis of a distributions. In contrast to noise, outliers can be valuable to the analyst since they can indicate new trends in the data or the importance of unrecognised but still analysis influencing factors.

- **Systematic literature review of the empirical software engineering literature searching for evidence of data quality considerations**
  An exhaustive systematic literature review has been carried out investigating to what extent the empirical software engineering community has considered data quality. The findings have bee quantified and provided a snapshot of the state of the issue of data quality in empirical software engineering. The findings are:

  - 161 papers considered noise or data quality. This is a very small proportion (about 1%) of all papers published in the empirical software engineering domain. This shows that the issues of noise and data quality have been largely neglected by the community.

  - The most dominant approach (31% of all 161 retrieved papers) to deal with poor data quality was the improvement of data collection procedures. This often not possible since analysts in the empirical software engineering domain have to work with historical data and do not have influence on the data collection process [66]. Therefore it can only be a suggestion in many cases.
  - The most dominant practical approach (22% of all retrieved papers) was manual data quality checking. This is a costly process since it requires the full-time attention of a analyst.
  - 11% of all retrieved papers utilised data quality meta-data in order to identify and eliminate poor data quality instances from a data set, but these meta-data were surrogates for missingness only, and not for inaccuracy.
  - To fields largely neglected by the community are the application of automated noise handling, and empirical analyses of the data quality. The research into both issues is limited and at time not replicable.
  - None of the papers retrieved utilised data quality protocols. This is very problematic since it makes replicated studies difficult.

- **Conducted a multi-pronged approach to investigate the effectiveness of three noise handling techniques**
  Three complementary investigations have been carried out to research the effectiveness of three noise handling techniques. The first investigation focussed on predictive accuracy a measure widely used in the machine learning community. The results of this investigation were quite positive especially for the filtering and polish technique. The results of the following investigations contradicted the findings of the initial investigation for all techniques. This highlighted a potential weakness of the predictive accuracy measure. In retrospect this seems logical since it only measures how well a model built using a set of data predicts the outcomes of another set of data. Therefore the conclusion is that reliance on predictive accuracy solely to measure the effectiveness

of noise handling techniques to discover 'true' noise is insufficient, especially not in a domain where outliers are considered to be valuable instances.

- **Highlight possible shortcomings of current automated noise handling techniques**
  Whilst this thesis only investigated three noise handling techniques, their questionable effectiveness highlights a shortcoming of automated noise handling techniques. It is concluded that automated noise handling should be integrated into data quality procedures like the one suggested by Kimball [96], where the domain knowledge is a constant input into the data cleaning process. A silver bullet automated cleansing technique might not be possible.

7.3 Limitations and Possible Future Work

The work carried out for this thesis resulted in the ascertaining of several issues which might have limiting impact on the presented results and could pose threads to validity. Where appropriate future work opportunities are listed.

- **Multi-pronged approach to investigate the effectiveness of other noise handling techniques**
  This thesis has only investigated the effectiveness of noise handling techniques which utilised decision trees. Other machine learning techniques especially in the effort estimation domain which can incorporate the continuous nature of the dependent variable could be more effective. It should also be considered if these techniques could accommodate more domain knowledge such that they could be fine tuned for the noise cleaning task.

- **Inefficiencies of bibliographic databases**
  Some of the bibliographic databases used for the systematic literature review created problems for the literature search. The issues encountered were:

  - The user interfaces of the databases were not all user friendly. Especially the

ScienceDirect database had to incorporate iterative search refinement and could not deal with very long search strings.

  - The IEEE Explore database changed its user interface in between searches, thus forcing editing and customising the initial searches to accommodate the new search interface.

- **Search terms with many synonyms**

  Data quality has many different synonyms. Not all synonyms might have been found resulting in possibly missing some relevant papers. This issue was attempted to be overcome by carrying out a pilot search and searching through candidate papers in order to find possible search terms, but still not all possible synonyms might have been found.

- **Simplistic simulation**

  The simulated data sets used in this thesis were based on only one real world data set. A simple linear relationship between independent and dependent variables was assumed and used as underlying model. This might have aided the noise detection techniques, which makes the findings even more poignant. Future investigations could investigate different underlying models. Valid outliers could be modelled by introducing separate models for their creation.

Apart from the future research opportunities connected to the above listed limitations further research opportunities are:

- **Development of unified data quality protocols for the empirical software engineering community**

  The results of the systematic literature review show that none of the investigations used data quality protocols. Data cleaning should utilise the usage of protocols describing the data cleaning process. This would ensure rigour and repeatability of future empirical software engineering investigations. The development of a unified data quality protocol could help with easy identification of applied noise handling techniques and the identification of the noisy instances which then could be examined again by other researchers improving academic rigour.

- **More analyses of automated algorithms**

  More existing noise cleaning algorithms should be investigated, testing the algorithms' effectiveness in different environments and on a range of data sets. Different environments like data sets with occurrences of 'true' exceptional instances (outliers) with modelled underlying relationships could be created, and an algorithm's ability to detect noise rather than the introduced 'true' exceptional instances could be tested.

Bibliography

[1] Bram Adams, Zhen Ming Jiang, and Ahmed E. Hassan. Identifying cross-cutting concerns using historical code changes. In *ICSE '10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, pages 305–314, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-719-6. doi: http://doi.acm.org/10.1145/1806799.1806846.

[2] Moataz A. Ahmed and Zeeshan Muzaffar. Handling imprecision and uncertainty in software development effort prediction: A type-2 fuzzy logic based framework. *Inf. Softw. Technol.*, 51(3):640–654, 2009. ISSN 0950-5849. doi: http://dx.doi.org/10.1016/j.infsof.2008.09.004.

[3] Maurcio Amaral De Almeida and Stan Matwin. Machine learning method for software quality model building. In *Proceedings of the Eleventh International Symposium on Methodologies for Intelligent Systems (ISMIS*, pages 565–573, 1999.

[4] James H. Andrews and Tim Menzies. On the value of combining feature subset selection with genetic algorithms: faster learning of coverage models. In *PROMISE '09: Proceedings of the 5th International Conference on Predictor Models in Software Engineering*, pages 1–10, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-634-2. doi: http://doi.acm.org/10.1145/1540438.1540456.

[5] Oliver Arafat and Dirk Riehle. The commenting practice of open source. In *OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, pages

857–864, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-768-4. doi: http://doi.acm.org/10.1145/1639950.1640047.

[6] Jorge Aranda and Gina Venolia. The secret life of bugs: Going past the errors and omissions in software repositories. In *ICSE '09: Proceedings of the 31st International Conference on Software Engineering*, pages 298–308, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-1-4244-3453-4.

[7] Erik Arisholm. Empirical assessment of the impact of structural properties on the changeability of object-oriented software. *Information and Software Technology*, 48 (11):1046–1055, 2006.

[8] Phillip G. Armour. Software: hard data. *Communication of the ACM*, 49(9):15–17, 2006. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/1151030.1151043.

[9] Yoris A. Au, Darrell Carpenter, Xiaogang Chen, and Jan G. Clark. Virtual organizational learning in open source software development projects. *Inf. Manage.*, 46(1): 9–15, 2008. ISSN 0378-7206. doi: http://dx.doi.org/10.1016/j.im.2008.09.004.

[10] Martin Auer and Stefan Biffl. Increasing the accuracy and reliability of analogy-based cost estimation with extensive project feature dimension weighting. In *ISESE '04: Proceedings of the 2004 International Symposium on Empirical Software Engineering*, pages 147–155, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2165-7. doi: http://dx.doi.org/10.1109/ISESE.2004.22.

[11] Muhammad Ali Babar and He Zhang. Systematic literature reviews in software engineering: Preliminary results from interviews with researchers. In *ESEM '09: Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 346–355, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-1-4244-4842-5. doi: http://dx.doi.org/10.1109/ESEM.2009.5314235.

[12] Adrian Bachmann and Abraham Bernstein. Software process data quality and characteristics: a historical view on open and closed source projects. In *IWPSE-Evol '09:*

Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops, pages 119–128, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-678-6.

[13] Sohaib Shahid Bajwa and Cigdem Gencel. What are the significant cost drivers for cosmic functional size based effort estimation? In *IWSM '09 /Mensura '09: Proceedings of the International Conferences on Software Process and Product Measurement*, pages 62–75, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-05414-3.

[14] K. Bennett, E. Burd, C. Kemerer, M. M. Lehman, M. Lee, R. Madachy, C. Mair, D. Sjoberg, and S. Slaughter. Empirical studies of evolving systems. *Empirical Software Engineering*, 4(4):370–380, 1999. ISSN 1382-3256. doi: http://dx.doi.org/10.1023/A:1009869705323.

[15] Stanislav Berlin, Tzvi Raz, Chanan Glezer, and Moshe Zviran. Comparison of estimation methods of cost and duration in it projects. *Inf. Softw. Technol.*, 51(4):738–748, 2009. ISSN 0950-5849.

[16] Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann. What makes a good bug report? In *SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 308–318, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-995-1. doi: http://doi.acm.org/10.1145/1453101.1453146.

[17] S. Bibi, I. Stamelos, and L. Angelis. Combining probabilistic models for explanatory productivity estimation. *Inf. Softw. Technol.*, 50(7-8):656–669, 2008. ISSN 0950-5849. doi: http://dx.doi.org/10.1016/j.infsof.2007.06.004.

[18] S. Bibi, G. Tsoumakas, I. Stamelos, and I. Vlahavas. Regression via classification applied on software defect estimation. *Expert Syst. Appl.*, 34(3):2091–2101, 2008. ISSN 0957-4174. doi: http://dx.doi.org/10.1016/j.eswa.2007.02.012.

[19] Stefan Biffl and Walter J. Gutjahr. Using a reliability growth model to control software

inspection. *Empirical Software Engineering*, 7(3):257–284, 2002. ISSN 1382-3256. doi: http://dx.doi.org/10.1023/A:1016396232448.

[20] Christian Bird, Adrian Bachmann, Eirik Aune, John Duffy, Abraham Bernstein, Vladimir Filkov, and Premkumar Devanbu. Fair and balanced?: bias in bug-fix datasets. In *ESEC/FSE '09: Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 121–130, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-001-2.

[21] Barry W. Boehm and Kevin J. Sullivan. Software economics: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 319–343, New York, NY, USA, 2000. ACM. ISBN 1-58113-253-0. doi: http://doi.acm.org/10.1145/336512.336584.

[22] Grant Braught, Craig S. Miller, and David Reed. Core empirical concepts and skills for computer science. *SIGCSE Bull.*, 36(1):245–249, 2004. ISSN 0097-8418. doi: http://doi.acm.org/10.1145/1028174.971388.

[23] Carla E. Brodley and Mark A. Friedl. Identifying and eliminating mislabeled training instances. In *AAAI/IAAI, Vol. 1*, pages 799–805, 1996. URL `citeseer.ist.psu.edu/brodley96identifying.html`.

[24] Carla E. Brodley and Mark A. Friedl. Improving automated land cover mapping by identifying and eliminating mislabeled observations from training data, 1996. URL `citeseer.ist.psu.edu/brodley96improving.html`.

[25] Carla E. Brodley and Mark A. Friedl. Identifying mislabeled training data. *Journal of Artificial Intelligence Research (JAIR)*, 11:131–167, 1999.

[26] R. Buehler and D. Griffin. Planning, personality, and prediction: The role of future focus in optimistic time predictions. *Organizational Behavior and Human Decision Processes*, 92:80–90, 2003. Both expts based on undergrad students. Tasks were to

predict Xmas shopping completion and completing coursework. Both expts give strong confirmation of the planning fallacy ie future planning increases optimism and therefore bias. This is not moderated either by personality or time framing (predicting using an absolute date or deadline minus n days) or past reflections. It also generalised across 2 different problem domains.

[27] Luigi Buglione and Cigdem Gencel. Impact of base functional component types on software functional size based effort estimation. In *PROFES '08: Proceedings of the 9th international conference on Product-Focused Software Process Improvement*, pages 75–89, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-69564-6.

[28] Cagatay Catal and Banu Diri. Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. *Inf. Sci.*, 179 (8):1040–1058, 2009. ISSN 0020-0255.

[29] Jr-Shian Chen and Ching-Hsue Cheng. Software diagnosis using fuzzified attribute base on modified mepa. In Moonis Ali and Richard Dapoigny, editors, *Advances in Applied Artificial Intelligence, 19th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2006, Annecy, France, June 27-30, 2006, Proceedings*, volume 4031 of *Lecture Notes in Computer Science*, pages 1270–1279. Springer, 2006.

[30] Zhihao Chen, Barry Boehm, Tim Menzies, and Daniel Port. Finding the right data for software cost modeling. *IEEE Softw.*, 22(6):38–46, 2005. ISSN 0740-7459. doi: http://dx.doi.org/10.1109/MS.2005.151.

[31] Peter Clark and Tim Niblett. The cn2 induction algorithm. *Mach. Learn.*, 3(4):261–283, 1989. ISSN 0885-6125. doi: http://dx.doi.org/10.1023/A:1022641700528.

[32] Alberto Colombo, Ernesto Damiani, and Gabriele Gianini. Discovering the software process by means of stochastic workflow analysis. *Journal of Systems Architecture*, 52(11):684–692, 2006. ISSN 1383-7621. doi: http://dx.doi.org/10.1016/j.sysarc.2006.06.012.

[33] Steve Counsell, George Loizou, and Rajaa Najjar. Quality of manual data collection in java software: an empirical investigation. *Empirical Software Engineering*, 12(3): 275–293, 2006. ISSN 1382-3256. doi: http://dx.doi.org/10.1007/s10664-006-9028-y.

[34] Philip B. Crosby. *Quality without tears: The art of hassle free management.* McGraw-Hill, New York, USA, 1984. ISBN 007014530X.

[35] Juan J. Cuadrado-Gallego, Miguel Garre, Ricardo J. Rejas, and Miguel-Ángel Sicilia. Analysis of software functional size databases. pages 195–202, 2008.

[36] Juan J. Cuadrado-Gallego, Luigi Buglione, María J. Domínguez-Alda, Marian Fernández de Sevilla, J. Antonio Gutierrez de Mesa, and Onur Demirors. An experimental study on the conversion between ifpug and cosmic functional size measurement units. *Inf. Softw. Technol.*, 52(3):347–357, 2010. ISSN 0950-5849.

[37] Richard D. De Veaux and David J. Hand. How to lie with bad data. *Statistical Science*, 20(3):231–238, 2005.

[38] Prem Devanbu, Sakke Karstu, Walcélio Melo, and William Thomas. Analytical and empirical evaluation of software reuse metrics. In *ICSE '96: Proceedings of the 18th international conference on Software engineering*, pages 189–199, Washington, DC, USA, 1996. IEEE Computer Society. ISBN 0-8186-7246-3.

[39] E. Dillon and Christophe Meudec. Automatic test data generation from embedded c code. In *SAFECOMP*, pages 180–194, 2004.

[40] Liping Ding, Qiusong Yang, Liang Sun, Jie Tong, and Yongji Wang. Evaluation of the capability of personal software process based on data envelopment analysis. In Mingshu Li, Barry W. Boehm, and Leon J. Osterweil, editors, *Unifying the Software Process Spectrum, International Software Process Workshop, SPW 2005, Beijing, China, May 25-27, 2005, Revised Selected Papers*, volume 3840 of *Lecture Notes in Computer Science*, pages 235–248. Springer, 2005.

[41] Anne M. Disney and Phillip M. Johnson. Investigating data quality problems in the psp. *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 143–152, 1998. URL `www.scopus.com`. Cited By (since 1996): 2.

[42] R. Geoff Dromey. Software quality—prevention versus cure? *Software Quality Control*, 11(3):197–210, 2003. ISSN 0963-9314. doi: http://dx.doi.org/10.1023/A:1025162610079.

[43] Roland Ducournau, Floréal Morandat, and Jean Privat. Empirical assessment of object-oriented implementations with multiple inheritance and static typing. In *OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference on Object oriented programming systems languages and applications*, pages 41–60, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-766-0.

[44] Christof Ebert. Experiences with criticality predictions in software development. *SIGSOFT Softw. Eng. Notes*, 22(6):278–293, 1997. ISSN 0163-5948. doi: http://doi.acm.org/10.1145/267896.267916.

[45] Christof Ebert. Technical controlling and software process improvement. *Journal of Systems and Software*, 46(1):25–39, 1999. ISSN 0164-1212. doi: http://dx.doi.org/10.1016/S0164-1212(98)10086-9.

[46] Christof Ebert, Thomas Liedtke, and Ekkehard Baisch. Improving reliability of large software systems. *Ann. Softw. Eng.*, 8(1-4):3–51, 1999. ISSN 1022-7091. doi: http://dx.doi.org/10.1023/A:1018971212809.

[47] Jason B. Ellis, Shahtab Wahid, Catalina Danis, and Wendy A. Kellogg. Task and social visualization in software development: evaluation of a prototype. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 577–586, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-593-9. doi: http://doi.acm.org/10.1145/1240624.1240716.

[48] Norman Fenton, Martin Neil, William Marsh, Peter Hearty, Lukasz Radlinski, and

Paul Krause. Project data incorporating qualitative factors for improved software defect prediction. In *ICSEW '07: Proceedings of the 29th International Conference on Software Engineering Workshops*, page 69, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2830-9. doi: http://dx.doi.org/10.1109/ICSEW.2007.171.

[49] Norman Fenton, Martin Neil, William Marsh, Peter Hearty, Lukasz Radliński, and Paul Krause. On the effectiveness of early life cycle defect prediction with bayesian nets. *Empirical Softw. Engg.*, 13(5):499–537, 2008. ISSN 1382-3256. doi: http://dx.doi.org/10.1007/s10664-008-9072-x.

[50] Norman E. Fenton and Martin Neil. A critique of software defect prediction models. *IEEE Transactions on Software Engineering*, 25(5):675–689, 1999. ISSN 0098-5589. doi: http://dx.doi.org/10.1109/32.815326.

[51] Norman E. Fenton and Shari L. Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. PWS, 2nd edition, 1996.

[52] Xiaoli Fern, Chaitanya Komireddy, Valentina Grigoreanu, and Margaret Burnett. Mining problem-solving strategies from hci data. *ACM Trans. Comput.-Hum. Interact.*, 17(1):1–22, 2010. ISSN 1073-0516.

[53] G. R. Finnie, G. E. Wittig, and J.-M. Desharnais. A comparison of software effort estimation techniques: using function points with neural networks, case-based reasoning and regression models. *Journal of Systems and Software*, 39(3):281–289, 1997. ISSN 0164-1212. doi: http://dx.doi.org/10.1016/S0164-1212(97)00055-1.

[54] G. R. Finnie, G. E. Wittig, and J. M. Desharnais. Estimating software development effort with case-based reasoning. In *ICCBR '97: Proceedings of the Second International Conference on Case-Based Reasoning Research and Development*, pages 13–22, London, UK, 1997. Springer-Verlag. ISBN 3-540-63233-6.

[55] Gavin R. Finnie and Gerhard E. Wittig. Ai tools for software development effort estimation. In *SEEP '96: Proceedings of the 1996 International Conference on Software*

Engineering: Education and Practice (SE:EP '96), page 346, Washington, DC, USA, 1996. IEEE Computer Society. ISBN 0-8186-7379-6.

[56] Andres Folleco, Taghi Khoshgoftaar, Jason Van Hulse, and Lofton A. Bullard. Software quality modeling: The impact of class noise on the random forest classifier. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2008, June 1-6, 2008, Hong Kong, China*, pages 3853–3859. IEEE, 2008.

[57] Dragan Gamberger and Nada Lavrač. Noise detection and elimination applied to noise handling in a krk chess endgame. In *ILP '96: Selected Papers from the 6th International Workshop on Inductive Logic Programming*, pages 72–88, London, UK, 1997. Springer-Verlag. ISBN 3-540-63494-0.

[58] Dragan Gamberger, Nada Lavrač, and Sašo Džeroski. Noise elimination in inductive concept learning: a case study in medical diagnosis. In *Algorithmic Learning Theory, 7th International Workshop, ALT '96, Sydney, Australia, October 1996, Proceedings*, volume 1160, pages 199–212. Springer, 1996. URL `citeseer.ist.psu.edu/article/gamberger96noise.html`.

[59] Dragan Gamberger, Nada Lavrač, and Ciril Grošelj. Experiments with noise detection algorithms in the diagnosis of coronary artery disease. In *IDAMAP-98, Third Workshop on Intelligent Data Analysis in Medicine and Pharmacology*, pages 29–33, Brighton, UK, 1998. University of Brighton. URL `citeseer.ist.psu.edu/224846.html`.

[60] Dragan Gamberger, Nada Lavrač, and Ciril Grošelj. Experiments with noise filtering in a medical domain. In *Proceedings of the 16th International Conference on Machine Learning*, pages 143–151. Morgan Kaufmann, San Francisco, CA, 1999. URL `citeseer.ist.psu.edu/article/gamberger99experiments.html`.

[61] Markus M. Geipel and Frank Schweitzer. Software change dynamics: evidence from 35 java projects. In *ESEC/FSE '09: Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The*

foundations of software engineering, pages 269–272, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-001-2.

[62] Cigdem Gencel and Luigi Buglione. Do base functional component types affect the relationship between software functional size and effort? pages 72–85, 2008.

[63] Michael Gertz, M. Tamer Özsu, Gunter Saake, and Kai-Uwe Sattler. Report on the Dagstuhl seminar: "Data quality on the web". *SIGMOD Record*, 33(1):127–132, 2004. ISSN 0163-5808. doi: http://doi.acm.org/10.1145/974121.974144.

[64] María Paula González, Jesús Lorés, and Antoni Granollers. Enhancing usability testing through datamining techniques: A novel approach to detecting usability problem patterns for a context of use. *Inf. Softw. Technol.*, 50(6):547–568, 2008. ISSN 0950-5849. doi: http://dx.doi.org/10.1016/j.infsof.2007.06.001.

[65] The PROMISE Group. Promise data. Available: http://promisedata.org/, Last accessed 21 July, 2010.

[66] Ronald Gulezian. Software quality measurement and modeling, maturity, control and improvement. *Proceedings of the IEEE International Software Engineering Standards Symposium*, pages 52–59, 1995. URL `www.scopus.com`.

[67] Volkmar H. Haase. Software process improvement planning with neural networks. In *EUROMICRO '98: Proceedings of the 24th Conference on EUROMICRO*, page 20808, Washington, DC, USA, 1998. IEEE Computer Society. ISBN 0-8186-8646-4-2.

[68] Stuart Hansen and Erica Eddy. Engagement and frustration in programming projects. In *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education*, pages 271–275, New York, NY, USA, 2007. ACM. ISBN 1-59593-361-1. doi: http://doi.acm.org/10.1145/1227310.1227407.

[69] Martin Hirzel. Data layouts for object-oriented programs. *SIGMETRICS Perform. Eval. Rev.*, 35(1):265–276, 2007. ISSN 0163-5999. doi: http://doi.acm.org/10.1145/1269899.1254915.

[70] Ray Horak. *Webster's New World Telecom Dictionary*. 2007. ISBN 047177457X.

[71] HP. Eds is now hp enterprise services, July 2010.

[72] Sun-Jen Huang and Nan-Hsing Chiu. Optimization of analogy weights by genetic algorithm for software effort estimation. *Information & Software Technology*, 48(11): 1034–1045, 2006.

[73] Sun-Jen Huang, Nan-Hsing Chiu, and Yu-Jen Liu. A comparative evaluation on the accuracies of software effort estimates from clustered data. *Inf. Softw. Technol.*, 50(9-10): 879–888, 2008. ISSN 0950-5849. doi: http://dx.doi.org/10.1016/j.infsof.2008.02.005.

[74] IFPUG. Ifpug: The international function point users group, July 2010.

[75] International Software Benchmarking Standards Group. Isbsg, July 2010.

[76] Shamsi T. Iqbal and Brian P. Bailey. Understanding and developing models for detecting and differentiating breakpoints during interactive tasks. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 697–706, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-593-9. doi: http://doi.acm.org/10.1145/1240624.1240732.

[77] Daniel R. Jeske and Xuemei Zhang. Some successful approaches to software reliability modeling in industry. *Journal of Systems and Software*, 74(1):85–99, 2005. ISSN 0164-1212. doi: http://dx.doi.org/10.1016/j.jss.2003.10.024.

[78] George H. John. Robust decision trees: Removing outliers from databases, 1995. URL `citeseer.comp.nus.edu.sg/john95robust.html`.

[79] Philip Johnson and Shaoxuan Zhang. We need more coverage, stat! classroom experience with the software icu. In *ESEM '09: Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 168–178, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-1-4244-4842-5.

[80] Philip M. Johnson. Reengineering inspection. *Communications of the ACM*, 41(2): 49–52, 1998. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/269012.269020.

[81] Philip M. Johnson. Leap: a "personal information environment" for software engineers. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 654–657, New York, NY, USA, 1999. ACM. ISBN 1-58113-074-0. doi: http://doi.acm.org/10.1145/302405.302919.

[82] Phillip M. Johnson and Anne M. Disney. Personal software process: A cautionary case study. *IEEE Software*, 15(6):85–88, 1998.

[83] Phillip M. Johnson and Anne M. Disney. A critical analysis of psp data quality: Results from a case study. *Empirical Software Engineering*, 4(4):317–349, 1999.

[84] Sachin Katti, Balachander Krishnamurthy, and Dina Katabi. Collaborating against common enemies. In *IMC '05: Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, pages 34–34, Berkeley, CA, USA, 2005. USENIX Association.

[85] Taghi Khoshgoftaar and Edward B. Allen. Ordering fault-prone software modules. *Software Quality Control*, 11(1):19–37, 2003. ISSN 0963-9314. doi: http://dx.doi.org/10.1023/A:1023632027907.

[86] Taghi Khoshgoftaar and Pierre Rebours. Generating multiple noise elimination filters with the ensemble-partitioning filter. In *Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration*, pages 369–375, Las Vegas, NV, November 2004.

[87] Taghi Khoshgoftaar and Pierre Rebours. Improving software quality prediction by noise filtering techniques. *Journal of Computer Science and Technology*, 22(3):387–396, 2007.

[88] Taghi Khoshgoftaar and Naeem Seliya. The necessity of assuring quality in software measurement data. In *METRICS '04: Proceedings of the Software Metrics, 10th In-*

ternational Symposium, pages 119–130, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2129-0. doi: http://dx.doi.org/10.1109/METRICS.2004.41.

[89] Taghi Khoshgoftaar and Jason Van Hulse. Identifying noise in an attribute of interest. In *ICMLA '05: Proceedings of the Fourth International Conference on Machine Learning and Applications (ICMLA'05)*, pages 55–62, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2495-8. doi: http://dx.doi.org/10.1109/ICMLA.2005.39.

[90] Taghi Khoshgoftaar and Jason Van Hulse. Imputation techniques for multivariate missingness in software measurement data. *Software Quality Control*, 16(4):563–600, 2008. ISSN 0963-9314. doi: http://dx.doi.org/10.1007/s11219-008-9054-7.

[91] Taghi Khoshgoftaar, Naeem Seliya, and Kehan Gao. Rule-based noise detection for software measurement data. In *IRI*, pages 302–307, 2004.

[92] Taghi Khoshgoftaar, Andres Folleco, Jason Van Hulse, and Lofton A. Bullard. Software quality imputation in the presence of noisy data. In *Proceedings of the 2006 IEEE International Conference on Information Reuse and Integration, IRI - 2006: Heuristic Systems Engineering, September 16-18, 2006, Waikoloa, Hawaii, USA*, pages 484–489. IEEE Systems, Man, and Cybernetics Society, 2006.

[93] Taghi Khoshgoftaar, Angela Herzberg, and Naeem Seliya. Resource oriented selection of rule-based classification models: An empirical case study. *Software Quality Control*, 14(4):309–338, 2006. ISSN 0963-9314. doi: http://dx.doi.org/10.1007/s11219-006-0038-1.

[94] Taghi Khoshgoftaar, Naeem Seliya, and Nandini Sundaresh. An empirical study of predicting software faults with case-based reasoning. *Software Quality Control*, 14(2): 85–111, 2006. ISSN 0963-9314. doi: http://dx.doi.org/10.1007/s11219-006-7597-z.

[95] Taghi M. Khoshgoftaar and Edward B. Allen. A comparative study of ordering and classification of fault-pronesoftware modules. *Empirical Softw. Engg.*, 4(2):159–186, 1999. ISSN 1382-3256. doi: http://dx.doi.org/10.1023/A:1009876418873.

[96] Ralph Kimball. Dealing with dirty data. *DBMS*, 9(10):55–60, 1996. ISSN 1041-5173.

[97] B. A. Kitchenham. Empirical studies of assumptions that underlie software cost-estimation models. *Inf. Softw. Technol.*, 34(4):211–218, 1992. ISSN 0950-5849. doi: http://dx.doi.org/10.1016/0950-5849(92)90077-3.

[98] Barbara Kitchenham. Procedures for performing systematic reviews (technical report tr/se-0401). Technical Report Technical Report TR/SE-0401, Keele University, Keele, UK, July 2004.

[99] A. Güneş Koru and Jeff Tian. Defect handling in medium and large open source projects. *IEEE Softw.*, 21(4):54–61, 2004. ISSN 0740-7459. doi: http://dx.doi.org/10.1109/MS.2004.12.

[100] A. Güneş Koru, Khaled El Emam, Dongsong Zhang, Hongfang Liu, and Divya Mathew. Theory of relative defect proneness. *Empirical Softw. Engg.*, 13(5):473–498, 2008. ISSN 1382-3256. doi: http://dx.doi.org/10.1007/s10664-008-9080-x.

[101] J. R. Landis and G. G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159–174, March 1977. ISSN 0006-341X. URL `http://view.ncbi.nlm.nih.gov/pubmed/843571`.

[102] Jouni Lappalainen. Tool support for personal software process. In Frank Bomarius and Seija Komi-Sirviö, editors, *Product Focused Software Process Improvement, 6th International Conference, PROFES 2005, Oulu, Finland, June 13-15, 2005, Proceedings*, volume 3547 of *Lecture Notes in Computer Science*, pages 545–559. Springer, 2005.

[103] Luigi Lavazza. Convertibility of functional size measurements: new insights and methodological issues. In *PROMISE '09: Proceedings of the 5th International Conference on Predictor Models in Software Engineering*, pages 1–12, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-634-2. doi: http://doi.acm.org/10.1145/1540438.1540451.

[104] Meir M. Lehman and Juan F. Ramil. Software evolution and software evolution processes. *Ann. Softw. Eng.*, 14(1-4):275–309, 2002. ISSN 1022-7091. doi: http://dx.doi.org/10.1023/A:1020557525901.

[105] Jingyue Li, Finn Olav Bjrnson, Reidar Conradi, and Vigdis B. Kampenes. An empirical study of variations in cots-based software development processes in the norwegian it industry. *Empirical Software Engineering*, 11(3):433–461, 2006. ISSN 1382-3256. doi: http://dx.doi.org/10.1007/s10664-006-9005-5.

[106] Naixin Li and Yashwant K. Malaiya. Enhancing accuracy of software reliability prediction. In *4th International Symposium on Software Reliability Engineering*, pages 71–79, Denver, November 1993.

[107] Y. F. Li, M. Xie, and T. N. Goh. A study of project selection and feature weighting for analogy based software cost estimation. *Journal of Systems and Software*, 82(2): 241–252, 2009. ISSN 0164-1212. doi: http://dx.doi.org/10.1016/j.jss.2008.06.001.

[108] Gernot A. Liebchen and Martin Shepperd. Software productivity analysis of a large data set and issues of confidentiality and data quality. *Proceedings of the 11th IEEE International Software Metrics Symposium (METRICS'05)*, 00:46, 2005. ISSN 1530-1435. doi: http://doi.ieeecomputersociety.org/10.1109/METRICS.2005.43.

[109] Gernot A. Liebchen and Martin Shepperd. Data sets and data quality in software engineering. In *PROMISE '08: Proceedings of the 4th international workshop on Predictor models in software engineering*, pages 39–44, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-036-4. doi: http://doi.acm.org/10.1145/1370788.1370799.

[110] Gernot A. Liebchen, Bheki Twala, Martin Shepperd, and Michelle Cartwright. Assessing the quality and cleaning of a software project data set: An experience report. In *Proceedings of 10th International Conference on Evaluation and Assessment in Software Engineering (EASE).* British Computer Society, 2006.

[111] Gernot A. Liebchen, Bheki Twala, Martin Shepperd, Michelle Cartwright, and Mark

Stephens. Filtering, robust filtering, polishing: Techniques for addressing quality in software data. *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, 0:99–106, 2007. ISSN 1938-6451. doi: http://doi.ieeecomputersociety.org/10.1109/ESEM.2007.48.

[112] Roderick J. A. Little and Donald B. Rubin. *Statistical analysis with missing data.* John Wiley & Sons, Inc., New York, NY, USA, 1986. ISBN 0-471-80254-9.

[113] Chris Lokan and Emilia Mendes. Cross-company and single-company effort models using the isbsg database: a further replicated study. In *ISESE '06: Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, pages 75–84, New York, NY, USA, 2006. ACM. ISBN 1-59593-218-6. doi: http://doi.acm.org/10.1145/1159733.1159747.

[114] Chris Lokan and Emilia Mendes. Applying moving windows to software effort estimation. In *ESEM '09: Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 111–122, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-1-4244-4842-5.

[115] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[116] Jonathan I. Maletic and Andrian Marcus. Data cleansing: Beyond integrity analysis. In *Fifth Conference on Information Quality (IQ 2000)*, pages 200–209. MIT, 2000.

[117] Jonathan I. Maletic and Andrian Marcus. Data cleansing - a prelude to knowledge discovery. In Oded Maimon and Lior Rokach, editors, *The Data Mining and Knowledge Discovery Handbook*, pages 21–36. Springer, 2005. ISBN 0-387-24435-2.

[118] Michel Manago and Yves Kodratoff. Noise and knowledge acquisition. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, pages 348–354, 1987.

[119] T. Martinetz, S. Berkovich, and K. Schulten. “Neural-gas” Network for Vector Quantization and its Application to Time-Series Prediction. *IEEE-Transactions on Neural Networks*, 4(4):558–569, 1993.

[120] Emilia Mendes. The use of a bayesian network for web effort estimation. In *ICWE'07: Proceedings of the 7th international conference on Web engineering*, pages 90–104, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-73596-0.

[121] Emilia Mendes and Chris Lokan. Replicating studies on cross- vs single-company effort models using the isbsg database. *Empirical Software Engineering*, 13(1), 2008. ISSN 1382-3256. doi: 10.1007/s10664-007-9045-5.

[122] Emilia Mendes, Ian Watson, Chris Triggs, Nile Mosley, and Steve Counsell. A comparative study of cost estimation models for web hypermedia applications. *Empirical Software Engineering*, 8(2):163–196, 2003. ISSN 1382-3256. doi: http://dx.doi.org/10.1023/A:1023062629183.

[123] Emilia Mendes, Sergio Di Martino, Filomena Ferrucci, and Carmine Gravino. Effort estimation: how valuable is it for a web company to use a cross-company data set, compared to using its own single-company data set? In Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, editors, *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, pages 963–972. ACM, 2007.

[124] Merriam-Webster Online. Merriam-Webster Online, 2010. URL `http://www.merriam-webster.com`.

[125] Audris Mockus. Software support tools and experimental work. In *Proceedings of the 2006 international conference on Empirical software engineering issues*, pages 91–99, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-71300-5.

[126] Audris Mockus. Succession: Measuring transfer of code and developer productivity. In *ICSE '09: Proceedings of the 31st International Conference on Software Engineering*,

pages 67–77, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-1-4244-3453-4. doi: http://dx.doi.org/10.1109/ICSE.2009.5070509.

[127] Parastoo Mohagheghi and Reidar Conradi. Quality, productivity and economic benefits of software reuse: a review of industrial studies. *Empirical Software Engineering*, 12(5): 471–516, 2007. ISSN 1382-3256. doi: http://dx.doi.org/10.1007/s10664-007-9040-x.

[128] Sandro Morasca and Günther Ruhe. A hybrid approach to analyze empirical software engineering data and its application to predict module fault-proneness in maintenance. *Journal of Systems and Software*, 53(3):225–237, 2000. ISSN 0164-1212. doi: http://dx.doi.org/10.1016/S0164-1212(00)00014-5.

[129] Raimund Moser, Witold Pedrycz, and Giancarlo Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In Wilhelm Schäfer, Matthew B. Dwyer, and Volker Gruhn, editors, *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*, pages 181–190. ACM, 2008.

[130] J. Moses. Bayesian probability distributions for assessing measurement of subjective software attributes. *Information and Software Technology*, 42(8):533–546, 2000. ISSN 0950-5849. doi: DOI: 10.1016/S0950-5849(00)00097-5.

[131] Heiko Müller and Johann-Christoph Freytag. Problems, methods, and challenges in comprehensive data cleansing. Technical report, Humboldt University Berlin, 2003.

[132] John C. Munson and Allen P. Nikora. Toward a quantifiable definition of software faults. In *ISSRE '02: Proceedings of the 13th International Symposium on Software Reliability Engineering*, page 388, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-8186-1763-3.

[133] John C. Munson, Allen P. Nikora, and Joseph S. Sherif. Software faults: A quantifiable definition. *Advances in Engineering Software*, 37(5):327–333, 2006.

[134] Volker Nannen. The paradox of overfitting. Master's thesis, Rijksuniversiteit Groningen, the Netherlands, 2003.

[135] Richi Nayak and Tian Qiu. Use of data mining in system development life cycle. In Graham J. Williams and Simeon J. Simoff, editors, *Data Mining - Theory, Methodology, Techniques, and Applications*, volume 3755 of *Lecture Notes in Computer Science*, pages 105–117. Springer, 2006.

[136] C.L. Blake D.J. Newman and C.J. Merz. UCI Repository of machine learning databases, 1998. URL `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[137] Thomas J. Ostrand, Elaine J. Weyuker, and Robert M. Bell. Where the bugs are. In *ISSTA '04: Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*, pages 86–96, New York, NY, USA, 2004. ACM. ISBN 1-58113-820-2. doi: http://doi.acm.org/10.1145/1007512.1007524.

[138] Thomas J. Ostrand, Elaine J. Weyuker, and Robert M. Bell. Predicting the location and number of faults in large software systems. *IEEE Transactions in Software Engineering*, 31(4):340–355, 2005. ISSN 0098-5589. doi: http://dx.doi.org/10.1109/TSE.2005.49.

[139] Parag C. Pendharkar. An exploratory study of object-oriented software component size determinants and the application of regression tree forecasting models. *Inf. Manage.*, 42(1):61–73, 2004. ISSN 0378-7206. doi: http://dx.doi.org/10.1016/j.im.2003.12.004.

[140] Parag C. Pendharkar and James A. Rodger. The relationship between software development team size and software development cost. *Communication of the ACM*, 52(1): 141–144, 2009. ISSN 0001-0782.

[141] Parag C. Pendharkar, James A. Rodger, and Girish H. Subramanian. An empirical study of the cobb-douglas production function properties of software development effort. *Inf. Softw. Technol.*, 50(12):1181–1188, 2007. ISSN 0950-5849. doi: http://dx.doi.org/10.1016/j.infsof.2007.10.019.

[142] Yi Peng, Guoxun Wang, and Honggang Wang. User preferences based software defect detection algorithms selection using mcdm. *Information Sciences*, In Press, Corrected Proof:–, 2010. ISSN 0020-0255.

[143] Lesley Pickard, Barbara Kitchenham, and Stephen G. Linkman. Using simulated data sets to compare data analysis techniques used for software cost modelling. *IEE Proceedings - Software*, 148(6):165–174, 2001.

[144] Rahul Premraj, Martin Shepperd, Barbara Kitchenham, and Pekka Forselius. An empirical analysis of software productivity over time. In *METRICS '05: Proceedings of the 11th IEEE International Software Metrics Symposium*, page 37, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2371-4.

[145] Rossane Prince and Graeme G. Shanks. A semiotic information quality framework. In *Proceedings of IFIP International Conference on Decision Support Systems (DSS2004): Decision Support in an Uncertain and Complex World*, 2004.

[146] Radliski, N. E. Fenton, M Neil, and D Marquez. Modelling prior productivity and defect rates in a causal model for software project risk assessment. *Polish Journal of Environmental Studies*, 16(4A):256–260, 2007. Hard, Olsztyn.

[147] Thomas C. Redman. *Data Quality for the Information Age.* Artech House, Inc., Norwood, MA, USA, 1996. ISBN 0890068836. Foreword By-A. Blanton Godfrey.

[148] Thomas C. Redman. The impact of poor data quality on the typical enterprise. *Communications of the ACM*, 41(2):79–82, 1998. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/269012.269025.

[149] Romain Robbes and Michele Lanza. Spyware: a change-aware development toolset. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 847–850, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-079-1. doi: http://doi.acm.org/10.1145/1368088.1368219.

[150] Gregorio Robles, Jesus M. Gonzalez-Barahona, and Juan Julian Merelo. Beyond source code: the importance of other artifacts in software development (a case study). *Journal of Systems and Software*, 79(9):1233–1248, 2006. ISSN 0164-1212. doi: http://dx.doi.org/10.1016/j.jss.2006.02.048.

[151] Li Ruan, Yongji Wang, Qing Wang, Mingshu Li, Yun Yang, Lizi Xie, Dapeng Liu, Haitao Zeng, Shen Zhang, Junchao Xiao, Lei Zhang, M. Wasif Nisar, and Jian Dai. Empirical study on benchmarking software development tasks. In *ICSP'07: Proceedings of the 2007 international conference on Software process*, pages 221–232, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-72425-4.

[152] Vladimir Rubin, Christian W. Günther, Wil M. P. Van Der Aalst, Ekkart Kindler, Boudewijn F. Van Dongen, and Wilhelm Schäfer. Process mining framework for software processes. In *ICSP'07: Proceedings of the 2007 international conference on Software process*, pages 169–181, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-72425-4.

[153] Guenther Ruhe. Rough set based data analysis in goal oriented software measurement. In *METRICS '96: Proceedings of the 3rd International Symposium on Software Metrics*, page 10, Washington, DC, USA, 1996. IEEE Computer Society. ISBN 0-8186-7364-8.

[154] Holger Schackmann, Henning Schaefer, and Horst Lichter. Evaluating process quality based on change request data — an empirical study of the eclipse project. In *IWSM '09 /Mensura '09: Proceedings of the International Conferences on Software Process and Product Measurement*, pages 227–241, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-05414-3.

[155] Joost Schalken and Hans van Vliet. Measuring where it matters: Determining starting points for metrics collection. *Journal of Systems and Software*, 81(5):603–615, 2008. ISSN 0164-1212. doi: http://dx.doi.org/10.1016/j.jss.2007.07.041.

[156] Carolyn B. Seaman, Forrest Shull, Myrna Regardie, Denis Elbert, Raimund L. Feldmann, Yuepu Guo, and Sally Godfrey. Defect categorization: making use of a decade of widely varying historical data. In *ESEM '08: Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 149–157, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-971-5. doi: http://doi.acm.org/10.1145/1414004.1414030.

[157] Naeem Seliya and Taghi M. Khoshgoftaar. Software quality estimation with limited fault data: a semi-supervised learning perspective. *Software Quality Control*, 15(3): 327–344, 2007. ISSN 0963-9314. doi: http://dx.doi.org/10.1007/s11219-007-9013-8.

[158] Panagiotis Sentas and Lefteris Angelis. Categorical missing data imputation for software cost estimation by multinomial logistic regression. *Journal of Systems and Software*, 79(3):404–414, 2006. ISSN 0164-1212. doi: http://dx.doi.org/10.1016/j.jss.2005.02.026.

[159] Panagiotis Sentas, Lefteris Angelis, Ioannis Stamelos, and Georgios L. Bleris. Software productivity and effort prediction with ordinal regression. *Information & Software Technology*, 47(1):17–29, 2005.

[160] Panagiotis Sentas, Lefteris Angelis, and Ioannis Stamelos. A statistical framework for analyzing the duration of software projects. *Empirical Software Engineering*, 13(2): 147–184, 2008.

[161] J. Shepperd, J. Ouellette, and J. Fernandez. Abandoning unrealistic optimism: Performance estimates and the temporal proximity of self-relevant feedback. *Journal of Personality and Social Psychology*, 70(4):844–855, 1996. Possible that self-verification theory (opportunity to find out) causes a loss of optimism eg just before an exam. There is also research on defensive pessimism [prob not relevant for software projects?] Results show self-relevant feedback reduces optimism [durr!]. Also found low-self esteem participants displayed this effect more than high-self esteem.

[162] Martin Shepperd. Software project economics: a roadmap. In *FOSE '07: 2007 Future*

of Software Engineering, pages 304–315, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2829-5. doi: http://dx.doi.org/10.1109/FOSE.2007.23.

[163] Martin Shepperd and Gada Kadoda. Comparing software prediction techniques using simulation. *IEEE Transactions on Software Engineering*, 27(11):1014–1022, 2001. ISSN 0098-5589. doi: http://dx.doi.org/10.1109/32.965341.

[164] Miyoung Shin and Amrit L. Goel. Modeling software component criticality using a machine learning approach. In Tag Gon Kim, editor, *Artificial Intelligence and Simulation, 13th International Conference on AI, Simulation, and Planning in High Autonomy Systems, AIS 2004, Jeju Island, Korea, October 4-6, 2004, Revised Selected Papers*, volume 3397 of *Lecture Notes in Computer Science*, pages 440–448. Springer, 2005.

[165] Forrest Shull, Jeffrey Carver, and Guilherme H. Travassos. An empirical methodology for introducing software processes. In *ESEC/FSE-9: Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 288–296, New York, NY, USA, 2001. ACM. ISBN 1-58113-390-1. doi: http://doi.acm.org/10.1145/503209.503248.

[166] Akbar Siami Namin, James H. Andrews, and Duncan J. Murdoch. Sufficient mutation operators for measuring test effectiveness. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 351–360, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-079-1. doi: http://doi.acm.org/10.1145/1368088.1368136.

[167] Julilus Sim and Chris C. Wright. The kappa statistic in reliability studies: Use, interpretation, and sample size requirements. *Physical Therapy*, March 2005.

[168] Jan Sinschek, Andreas Sewe, and Mira Mezini. Vm performance evaluation with functional models: an optimist's outlook. In *VMIL '09: Proceedings of the Third Workshop on Virtual Machines and Intermediate Languages*, pages 1–2, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-874-2.

[169] Raymund Sison, David Diaz, Eliska Lam, Dennis Navarro, and Jessica Navarro. Personal software process (psp) assistant. In *APSEC '05: Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC'05)*, pages 687–696, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2465-6. doi: http://dx.doi.org/10.1109/APSEC.2005.87.

[170] Sulayman K. Sowe, Ioannis Stamelos, and Lefteris Angelis. Identifying knowledge brokers that yield software engineering knowledge in oss projects. *Information & Software Technology*, 48(11):1025–1033, 2006.

[171] Sulayman K. Sowe, Ioannis Stamelos, and Lefteris Angelis. Understanding knowledge sharing activities in free/open source software projects: An empirical study. *Journal of Systems and Software*, 81(3):431–446, 2008. ISSN 0164-1212. doi: http://dx.doi.org/10.1016/j.jss.2007.03.086.

[172] Witawas Srisa-an, Myra B. Cohen, Yu Shang, and Mithuna Soundararaj. A self-adjusting code cache manager to balance start-up time and memory usage. In *CGO '10: Proceedings of the 8th annual IEEE/ACM international symposium on Code generation and optimization*, pages 82–91, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-635-9.

[173] Ioannis Stamelos, Lefteris Angelis, Maurizio Morisio, Evaggelos Sakellaris, and George L. Bleris. Estimating the development cost of custom software. *Inf. Manage.*, 40(8):729–741, 2003. ISSN 0378-7206. doi: http://dx.doi.org/10.1016/S0378-7206(02)00099-X.

[174] Erik Stensrud, Tron Foss, Barbara Kitchenham, and Ingunn Myrtveit. A further empirical investigation of the relationship between mre and project size. *Empirical Software Engineering*, 8(2):139–161, 2003. ISSN 1382-3256. doi: http://dx.doi.org/10.1023/A:1023010612345.

[175] Diane M. Strong, Yang W. Lee, and Richard Y. Wang. Data quality in con-

text. *Communications of the ACM*, 40(5):103–110, 1997. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/253769.253804.

[176] Giancarlo Succi, Witold Pedrycz, Milorad Stefanovic, and Barbara Russo. An investigation on the occurrence of service requests in commercial software applications. *Empirical Software Engineering*, 8(2):197–215, 2003.

[177] Thomas Tan, Qi Li, Barry Boehm, Ye Yang, Mei He, and Ramin Moazeni. Productivity trends in incremental and iterative software development. In *ESEM '09: Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 1–10, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-1-4244-4842-5.

[178] Wei Tang and Taghi M. Khoshgoftaar. Noise identification with the k-means algorithm. In *ICTAI '04: Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence*, pages 373–378, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2236-X. doi: http://dx.doi.org/10.1109/ICTAI.2004.93.

[179] Choh M. Teng. Correcting noisy data. In *Proceedings of the 16th International Conference on Machine Learning (ICML 99)*, pages 239–248, San Mateo, California, USA, 1999. Morgan Kaufmann.

[180] Choh M. Teng. Evaluating noise correction. In *Proceedings of the 6th Pacific Rim International Conference on Artificial Intelligence*. Springer-Verlag, 2000.

[181] Choh M. Teng. Combining noise correction with feature selection. pages 340–349, 2003.

[182] Thomas Thelin, Håkan Petersson, Per Runeson, and Claes Wohlin. Applying sampling to improve software inspections. *Journal of Systems and Software*, 73(2):257–269, 2004. ISSN 0164-1212. doi: http://dx.doi.org/10.1016/S0164-1212(03)00249-8.

[183] Terry M. Therneau and Elizabeth J. Atkinson. An introduction to recursive partitioning using the rpart routines. Technical report, Mayo Foundation, 1997.

[184] Christopher Thomson and Mike Holcombe. Correctness of data mined from cvs. In *MSR '08: Proceedings of the 2008 international working conference on Mining software repositories*, pages 117–120, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-024-1. doi: http://doi.acm.org/10.1145/1370750.1370777.

[185] Dwayne Towell and Jason Denton. A software implementation progress model. In Luciano Baresi and Reiko Heckel, editors, *Fundamental Approaches to Software Engineering, 9th International Conference, FASE 2006, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 27-28, 2006, Proceedings*, volume 3922 of *Lecture Notes in Computer Science*, pages 93–106. Springer, 2006.

[186] Jos J. Trienekens, Rob J. Kusters, Michiel J. Van Genuchten, and Hans Aerts. Targets, drivers and metrics in software process improvement: Results of a survey in a multinational organization. *Software Quality Control*, 15(2):135–153, 2007. ISSN 0963-9314. doi: http://dx.doi.org/10.1007/s11219-006-9007-y.

[187] Sylvie Trudel and Alain Abran. Functional size measurement quality challenges for inexperienced measurers. In *IWSM '09 /Mensura '09: Proceedings of the International Conferences on Software Process and Product Measurement*, pages 157–169, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-05414-3.

[188] Medha Umarji and Carolyn Seaman. Gauging acceptance of software metrics: Comparing perspectives of managers and developers. In *ESEM '09: Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 236–247, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-1-4244-4842-5. doi: http://dx.doi.org/10.1109/ESEM.2009.5315999.

[189] Jason Van Hulse and Taghi Khoshgoftaar. A comprehensive empirical evaluation of missing value imputation in noisy software measurement data. *Journal of Systems and Software*, 2007. doi: 10.1016/j.jss.2007.07.043.

[190] Jason Van Hulse and Taghi Khoshgoftaar. Knowledge discovery from imbalanced and noisy data. *Data Knowl. Eng.*, 68(12):1513–1542, 2009. ISSN 0169-023X.

[191] Jason Van Hulse, Taghi Khoshgoftaar, and Haiying Huang. The pairwise attribute noise detection algorithm. *Knowledge and Information Systems*, 11(2):171–190, 2006. URL `www.scopus.com`.

[192] Jason Van Hulse, Taghi Khoshgoftaar, Chris Seiffert, and Lili Zhao. Noise correction using bayesian multiple imputation. In *Proceedings of the 2006 IEEE International Conference on Information Reuse and Integration, IRI - 2006: Heuristic Systems Engineering, September 16-18, 2006, Waikoloa, Hawaii, USA*, pages 478–483. IEEE Systems, Man, and Cybernetics Society, 2006.

[193] Yair Wand and Richard Y. Wang. Anchoring data quality dimensions in ontological foundations. *Communincations of the ACM*, 39(11):86–95, 1996. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/240455.240479.

[194] Richard Y. Wang, Henry B. Kon, and Stuart E. Madnick. Data quality requirements analysis and modeling. In *Proceedings of the Ninth International Conference on Data Engineering*, pages 670–677, Washington, DC, USA, 1993. IEEE Computer Society. ISBN 0-8186-3570-3.

[195] Richard Y. Wang, Veda C. Storey, and Christopher P. Firth. A framework for analysis of data quality research. *IEEE Transactions on Knowledge and Data Engineering*, 7 (4):623–640, 1995. ISSN 1041-4347. doi: http://dx.doi.org/10.1109/69.404034.

[196] Xiaobo Wang, Guanhui Lai, and Chao Liu. Recovering relationships between documentation and source code based on the characteristics of software engineering. *Electron. Notes Theor. Comput. Sci.*, 243:121–137, 2009. ISSN 1571-0661.

[197] Xiaoyin Wang, Lu Zhang, Tao Xie, John Anvik, and Jiasu Sun. An approach to detecting duplicate bug reports using natural language and execution information. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*,

pages 461–470, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-079-1. doi: http://doi.acm.org/10.1145/1368088.1368151.

[198] Xiaoyin Wang, David Lo, Jing Jiang, Lu Zhang, and Hong Mei. Extracting paraphrases of technical terms from noisy parallel software corpora. In *ACL-IJCNLP '09: Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 197–200, Morristown, NJ, USA, 2009. Association for Computational Linguistics.

[199] Anders Wesslén. A replicated empirical study of the impact of the methods in the psp on individual engineers. *Empirical Software Engineering*, 5(2):93–123, 2000. ISSN 1382-3256. doi: http://dx.doi.org/10.1023/A:1009811222725.

[200] Isabella Wieczorek. Improved software cost estimation - a robust and interpretable modelling method and a comprehensive empirical investigation. *Empirical Software Engineering.*, 7(2):177–180, 2002. ISSN 1382-3256. doi: http://dx.doi.org/10.1023/A:1015206216560.

[201] Jeff Winter, Kari Rönkkö, and Mats Hellman. Reporting usability metrics experiences. In *CHASE '09: Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*, pages 108–115, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-1-4244-3712-2.

[202] Gerhard E. Wittig and Gavin R. Finnie. Estimating software development effort with connectionist models. *Information & Software Technology*, 39(7):469–476, 1997.

[203] Ye Yang, Mei He, Mingshu Li, Qing Wang, and Barry Boehm. Phase distribution of software development effort. In *ESEM '08: Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 61–69, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-971-5. doi: http://doi.acm.org/10.1145/1414004.1414016.

[204] Cemal Yilmaz, Amit Paradkar, and Clay Williams. Time will tell: fault localization using time spectra. In *ICSE '08: Proceedings of the 30th international conference on*

Software engineering, pages 81–90, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-079-1. doi: http://doi.acm.org/10.1145/1368088.1368100.

[205] Kyung-A Yoon and Doo-Hwan Bae. A pattern-based outlier detection method identifying abnormal attributes in software project data. *Inf. Softw. Technol.*, 52(2):137–151, 2010. ISSN 0950-5849.

[206] Du Zhang and Jeffrey J. P. Tsai. Machine learning and software engineering. *Software Quality Control*, 11(2):87–119, 2003. ISSN 0963-9314. doi: http://dx.doi.org/10.1023/A:1023760326768.

[207] Shen Zhang, Yongji Wang, Ye Yang, and Junchao Xiao. Capability assessment of individual software development processes using software repositories and dea. In *ICSP'08: Proceedings of the Software process, 2008 international conference on Making globally distributed software development a success story*, pages 147–159, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 3-540-79587-1, 978-3-540-79587-2.

[208] Zhenyu Zhang, Bo Jiang, W. K. Chan, T. H. Tse, and Xinming Wang. Fault localization through evaluation sequences. *Journal of Systems and Software*, 83(2):174–187, 2010. ISSN 0164-1212.

[209] Shi Zhong, Taghi M. Khoshgoftaar, and Naeem Seliya. Unsupervised learning for expert-based software quality estimation. In *HASE*, pages 149–155, 2004.

[210] Xingquan Zhu and Xindong Wu. Class noise vs. attribute noise: A quantitative study of their impacts. *Artificial Intelligence Review*, 22(3):177–210, 2004. ISSN 0269-2821. doi: http://dx.doi.org/10.1007/s10462-004-0751-8.

[211] Thomas Zimmermann and Peter Weigerber. Preprocessing cvs data for fine-grained analysis. In *Proceedings of the First International Workshop on Mining Software Repositories*, pages 2–6, May 2004.

[212] Thomas Zimmermann, Peter Weißgerber, Stephan Diehl, and Andreas Zeller. Mining version histories to guide software changes. In *ICSE '04: Proceedings of the 26th*

International Conference on Software Engineering, pages 563–572, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2163-0.

[213] Thomas Zimmermann, Valentin Dallmeier, Konstantin Halachev, and Andreas Zeller. erose: guiding programmers in eclipse. In *OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 186–187, New York, NY, USA, 2005. ACM. ISBN 1-59593-193-7. doi: http://doi.acm.org/10.1145/1094855.1094927.

Appendix A

Letter To PROMISE Program Committee

Dear *X*,

I am conducting a systematic review of the empirical software engineering literature on data quality. This is an extension of the paper published at PROMISE 2008 (and also attached), which resulted in a very interesting and encouraging discussion.

I am approaching you as an expert in this field of research and who has served on the program committee of PROMISE 2008.

In order to establish how effective my search (see below) of the literature has been, I would be grateful if you were able to identify one or more papers that you have authored or co-authored that you believe touch on issues of data quality (even if tangentially). I can then use these additional papers to test the thoroughness of my search protocol, and if necessary extend the search.

Whilst I appreciate how very busy you must be, a response (even a null response) would very much assist me in updating my literature review for my doctoral thesis which I hope to submit in the near future.

If you would like to receive a copy of the extended systematic review please let me know.

Many thanks in anticipation!

All the best,

Gernot Liebchen

CURRENT SEARCH PROTOCOL:

Search Terms: "data quality"AND software"

Searched Bibliographic Databases: ScienceDirect, SCOPUS and IEEExplore

Exhaustive Hand Search Of: Journal Empirical Software Engineering and the conference series of ESEM, METRICS, ISESE, PROMISE and EASE

Inclusion Criteria: The article must focus on an empirical investigation of some aspect of software engineering or address some methodological issue relevant to such empirical research. The article must address data quality explicitly. The article must be refereed. The article must be written in English.

Sub-Objectives: How significant do the community consider noise to be (in principle and in practice)? How do empirical analysts address this problem? Are there techniques that might be deployed to independently assess the quality of a given data set?

Last Search: January 2008

Appendix B

Paper Categorisations

Paper Categorisations *("Y" = Yes; Space = No; "?" = Not Stated)*

Paper Reference	Year	Quantitative	Qualitative	Data Collection	Manual Noise Checking	Automated Noise Checking	Empirical Analysis of Noise	Data Quality Meta Data	Is Noise A Problem?	Quality	Cost	Other
[106]	1993	Y							Y	Y		
[66]	1995	Y		Y	Y				Y			Y
[153]	1996	Y							Y		Y	
[55]	1996	Y							Y		Y	
[38]	1996	Y							Y		Y	
[44]	1997	Y							Y		Y	
[53]	1997	Y							Y		Y	
[54]	1997	Y							Y		Y	
[202]	1997	Y							Y		Y	
[67]	1998		Y	Y					Y			Y
[80]	1998	Y	Y	Y					Y			Y
[3]	1999	Y		Y					Y	Y		

Continued on the next page

Continued: Paper Categorisations *("Y" = Yes; Space = No; "?" = Not Stated)*

Paper Reference	Year	Quantitative	Qualitative	Data Collection	Manual Noise Checking	Automated Noise Checking	Empirical Analysis of Noise	Data Quality Meta Data	Is Noise A Problem?	Quality	Cost	Other
[95]	1999	Y	Y						Y	Y		
[46]	1999	Y							Y	Y		
[50]	1999	Y							Y	Y		
[45]	1999	Y		Y					Y		Y	
[14]	1999	?	?						?			Y
[81]	1999	Y	Y						Y			Y
[83]	1999		Y	Y	Y		Y		Y			Y
[128]	2000	Y							?	Y		
[21]	2000	Y							Y		Y	

Continued on the next page

Continued: Paper Categorisations *("Y" = Yes; Space = No; "?" = Not Stated)*												
Paper Reference	Year	Quantitative	Qualitative	Data Collection	Manual Noise Checking	Automated Noise Checking	Empirical Analysis of Noise	Data Quality Meta Data	Is Noise A Problem?	Quality	Cost	Other
[199]	2000	Y		Y								Y
[130]	2000	Y				Y			Y			Y
[165]	2001	Y	Y		Y				Y	Y		
[132]	2002	Y		Y					Y	Y		
[19]	2002		Y	Y	Y				Y	Y		
[200]	2002	Y		Y					?		Y	
[104]	2002		Y						Y			Y
[85]	2003	Y	Y						Y	Y		
[42]	2003		Y	Y					Y	Y		

Continued on the next page

Continued: Paper Categorisations *("Y" = Yes; Space = No; "?" = Not Stated)*

Paper Reference	Year	Quantitative	Qualitative	Data Collection	Manual Noise Checking	Automated Noise Checking	Empirical Analysis of Noise	Data Quality Meta Data	Is Noise A Problem?	Quality	Cost	Other
[173]	2003	Y		Y				Y	Y		Y	
[174]	2003	Y							Y		Y	
[122]	2003	Y							Y		Y	
[206]	2003	Y							Y			Y
[176]	2003	Y							Y			Y
[99]	2004	?	?	Y			Y		?	Y		
[209]	2004	Y				Y	Y		Y	Y		
[91]	2004	Y				Y	Y		Y	Y		
[178]	2004	Y				Y	Y		Y	Y		

Continued on the next page

Continued: Paper Categorisations *("Y" = Yes; Space = No; "?" = Not Stated)*

Paper Reference	Year	Quantitative	Qualitative	Data Collection	Manual Noise Checking	Automated Noise Checking	Empirical Analysis of Noise	Data Quality Meta Data	Is Noise A Problem?	Quality	Cost	Other
[86]	2004	Y				Y	Y		Y	Y		
[88]	2004	Y							Y	Y		
[39]	2004		Y						Y	Y		
[137]	2004		Y		Y				Y	Y		
[139]	2004	Y							?		Y	
[10]	2004	Y		Y					Y		Y	
[182]	2004		Y						Y			Y
[212]	2004	Y	Y						Y			Y
[211]	2004	Y	Y		Y				Y			Y

Continued on the next page

Continued: Paper Categorisations *("Y" = Yes; Space = No; "?" = Not Stated)*

Paper Reference	Year	Quantitative	Qualitative	Data Collection	Manual Noise Checking	Automated Noise Checking	Empirical Analysis of Noise	Data Quality Meta Data	Is Noise A Problem?	Quality	Cost	Other
[22]	2004	Y							Y			Y
[77]	2005	Y		Y					Y	Y		
[89]	2005	Y				Y	Y		Y	Y		
[164]	2005	Y							Y	Y		
[138]	2005		Y		Y				Y	Y		
[159]	2005	Y						Y	?		Y	
[30]	2005	Y	Y						Y		Y	
[108]	2005	Y			Y				Y		Y	
[40]	2005	Y							Y		Y	

Continued on the next page

Continued: Paper Categorisations *("Y" = Yes; Space = No; "?" = Not Stated)*

Paper Reference	Year	Quantitative	Qualitative	Data Collection	Manual Noise Checking	Automated Noise Checking	Empirical Analysis of Noise	Data Quality Meta Data	Is Noise A Problem?	Quality	Cost	Other
[213]	2005	Y	Y						Y			Y
[84]	2005	Y			Y				Y			Y
[102]	2005		Y	Y					Y			Y
[169]	2005		Y	Y					Y			Y
[29]	2006	Y							Y	Y		
[192]	2006	Y				Y	Y		Y	Y		
[92]	2006	Y			Y		Y		Y	Y		
[93]	2006	Y							Y	Y		
[94]	2006	Y							Y	Y		

Continued on the next page

Continued: Paper Categorisations *("Y" = Yes; Space = No; "?" = Not Stated)*												
Paper Reference	Year	Quantitative	Qualitative	Data Collection	Manual Noise Checking	Automated Noise Checking	Empirical Analysis of Noise	Data Quality Meta Data	Is Noise A Problem?	Quality	Cost	Other
[133]	2006	Y		Y					?	Y		
[191]	2006	Y				Y	Y		Y	Y		
[158]	2006	Y						Y	?		Y	
[113]	2006	Y						Y	Y		Y	
[110]	2006	Y			Y	Y	Y		Y		Y	
[7]	2006	Y		Y	Y				Y		Y	
[72]	2006	Y		Y					Y		Y	
[8]	2006	Y							Y			Y
[170]	2006		Y	Y	Y				Y			Y

Continued on the next page

Continued: Paper Categorisations *("Y" = Yes; Space = No; "?" = Not Stated)*

Paper Reference	Year	Quantitative	Qualitative	Data Collection	Manual Noise Checking	Automated Noise Checking	Empirical Analysis of Noise	Data Quality Meta Data	Is Noise A Problem?	Quality	Cost	Other
[185]	2006	Y		Y					Y			Y
[32]	2006	Y				Y			Y			Y
[150]	2006	Y		Y					?			Y
[135]	2006	?	?		Y				Y			Y
[33]	2006	Y		Y	Y		Y		Y			Y
[105]	2006	Y	Y	Y	Y				?			Y
[146]	2007	?	?					Y	?	Y	Y	
[151]	2007	Y		Y					?	Y	Y	
[87]	2007	Y				Y	Y		Y	Y		

Continued on the next page

Continued: Paper Categorisations *("Y" = Yes; Space = No; "?" = Not Stated)*

Paper Reference	Year	Quantitative	Qualitative	Data Collection	Manual Noise Checking	Automated Noise Checking	Empirical Analysis of Noise	Data Quality Meta Data	Is Noise A Problem?	Quality	Cost	Other
[48]	2007	Y	Y						?	Y		
[189]	2007	Y							Y	Y		
[157]	2007	Y				Y	Y		Y	Y		
[47]	2007		Y	Y					Y	Y		
[162]	2007	?	?						Y		Y	
[111]	2007	Y				Y	Y		Y		Y	
[141]	2007	Y						Y	?		Y	
[120]	2007	Y					Y		?		Y	
[123]	2007	?	?				Y		Y		Y	

Continued on the next page

Continued: Paper Categorisations *("Y" = Yes; Space = No; "?" = Not Stated)*

Paper Reference	Year	Quantitative	Qualitative	Data Collection	Manual Noise Checking	Automated Noise Checking	Empirical Analysis of Noise	Data Quality Meta Data	Is Noise A Problem?	Quality	Cost	Other
[76]	2007	Y			Y				Y			Y
[186]	2007	?	?	Y					?			Y
[68]	2007		Y	Y	Y				Y			Y
[69]	2007	Y		Y					?			Y
[152]	2007		Y	Y	?	Y			?			Y
[125]	2007	?	?	Y					Y			Y
[127]	2007	?	?					Y	Y			Y
[109]	2008	Y	Y						Y	Y	Y	Y
[100]	2008	Y		Y	Y				?	Y		

Continued on the next page

Continued: Paper Categorisations *("Y" = Yes; Space = No; "?" = Not Stated)*

Paper Reference	Year	Quantitative	Qualitative	Data Collection	Manual Noise Checking	Automated Noise Checking	Empirical Analysis of Noise	Data Quality Meta Data	Is Noise A Problem?	Quality	Cost	Other
[18]	2008	Y		Y				Y	Y	Y		
[56]	2008	Y							Y	Y		
[49]	2008	Y	Y	Y					Y	Y		
[9]	2008	Y		Y					Y	Y		
[90]	2008	Y							Y	Y		
[204]	2008	Y							Y	Y		
[166]	2008	Y							Y	Y		
[156]	2008		Y	Y					Y	Y		
[129]	2008	Y		Y					Y	Y		

Continued on the next page

Continued: Paper Categorisations (*"Y" = Yes; Space = No; "?" = Not Stated*)												
Paper Reference	Year	Quantitative	Qualitative	Data Collection	Manual Noise Checking	Automated Noise Checking	Empirical Analysis of Noise	Data Quality Meta Data	Is Noise A Problem?	Quality	Cost	Other
[35]	2008	Y			Y			Y	Y		Y	
[17]	2008	Y		Y				Y	Y		Y	
[27]	2008	Y						Y	Y		Y	
[62]	2008	Y						Y	Y		Y	
[73]	2008	Y						Y	Y		Y	
[203]	2008	Y		Y	Y				Y		Y	
[121]	2008	Y							?		Y	
[160]	2008	Y						Y	?		Y	
[16]	2008		Y		Y				Y			Y

Continued on the next page

Continued: Paper Categorisations *("Y" = Yes; Space = No; "?" = Not Stated)*

Paper Reference	Year	Quantitative	Qualitative	Data Collection	Manual Noise Checking	Automated Noise Checking	Empirical Analysis of Noise	Data Quality Meta Data	Is Noise A Problem?	Quality	Cost	Other
[197]	2008		Y						Y			Y
[184]	2008	Y	Y	Y	Y		Y		Y			Y
[207]	2008	Y	Y	Y	Y				Y			Y
[171]	2008		Y	Y	Y				Y			Y
[155]	2008	?	?		Y				?			Y
[149]	2008	?	?	Y					Y			Y
[64]	2008	Y							Y			Y
[4]	2009	Y							Y	Y		
[28]	2009	Y							Y	Y		

Continued on the next page

Continued: Paper Categorisations *("Y" = Yes; Space = No; "?" = Not Stated)*

Paper Reference	Year	Quantitative	Qualitative	Data Collection	Manual Noise Checking	Automated Noise Checking	Empirical Analysis of Noise	Data Quality Meta Data	Is Noise A Problem?	Quality	Cost	Other
[190]	2009	Y					Y		Y	Y		
[2]	2009	Y							Y		Y	
[13]	2009	Y						Y	Y		Y	
[15]	2009	Y							Y		Y	
[103]	2009	Y			Y				Y		Y	
[107]	2009	Y							Y		Y	
[114]	2009	Y						Y	Y		Y	
[126]	2009		Y		Y				Y		Y	
[140]	2009	Y						Y	Y		Y	

Continued on the next page

Continued: Paper Categorisations *("Y" = Yes; Space = No; "?" = Not Stated)*												
Paper Reference	Year	Quantitative	Qualitative	Data Collection	Manual Noise Checking	Automated Noise Checking	Empirical Analysis of Noise	Data Quality Meta Data	Is Noise A Problem?	Quality	Cost	Other
[177]	2009	Y			Y				Y		Y	
[187]	2009	Y			Y				Y		Y	
[5]	2009		Y	Y	Y				Y			Y
[6]	2009		Y	Y					Y			Y
[11]	2009		Y						Y			Y
[12]	2009		Y		Y		Y		Y			Y
[20]	2009	Y	Y						Y			Y
[43]	2009	Y		Y					Y			Y
[61]	2009		Y		Y				Y			Y

Continued on the next page

Continued: Paper Categorisations *("Y" = Yes; Space = No; "?" = Not Stated)*

Paper Reference	Year	Quantitative	Qualitative	Data Collection	Manual Noise Checking	Automated Noise Checking	Empirical Analysis of Noise	Data Quality Meta Data	Is Noise A Problem?	Quality	Cost	Other
[79]	2009		Y	Y					Y			Y
[154]	2009		Y						Y			Y
[168]	2009	Y							Y			Y
[188]	2009	Y							Y			Y
[198]	2009		Y						Y			Y
[201]	2009	Y							Y			Y
[205]	2010	Y				Y	Y		Y	Y	Y	
[142]	2010	Y			Y				Y	Y		
[208]	2010	Y							Y	Y		

Continued on the next page

Continued: Paper Categorisations *("Y" = Yes; Space = No; "?" = Not Stated)*

Paper Reference	Year	Quantitative	Qualitative	Data Collection	Manual Noise Checking	Automated Noise Checking	Empirical Analysis of Noise	Data Quality Meta Data	Is Noise A Problem?	Quality	Cost	Other
[36]	2010	Y		Y					Y		Y	
[1]	2010	Y							Y			Y
[52]	2010	Y	Y						Y			Y
[172]	2010	Y		Y					Y			Y
[196]	2010	Y			Y				Y			Y

Appendix C

Extracted Attributes from the EDS Database

Table 7.1: Innovation

Variable	Descriptive Name	Values	Comments
tech_innovation _Innovation_Desc	Technical Innovation	• "Previously done by work group" • "Done by others in EDS" • "Done by others in the industry" • "Industry prototype exists, not operational" • "New to industry, never done before"	An indicator for the technical innovation of the project.

Continued on the next page

Continued: Innovation			
bus_innovation _Innovation_Desc	Business Innovation	• "Previously done by work group" • "Done by others in EDS" • "Done by others in the industry" • "Industry prototype exists, not operational" • "New to industry, never done before"	An indicator for the business innovation of the project.
Continued on the next page			

Continued: Innovation			
appl_innovation _Innovation_Desc	Application Innovation	• "Previously done by work group" • "Done by others in EDS" • "Done by others in the industry" • "Industry prototype exists, not operational" • "New to industry, never done before"	An indicator of the degree if innovation of the application

Table 7.2: Concurrency, Team/Customer Complexity

Variable	Descriptive Name	Values	Comments
Concurrency_Desc	Concurrency	• "Not applicable" • "Hardware and Application" • "Hardware and Software" • "Application Code and System Software" • "Application Code, System Software, Hardware"	Indicates whether a project involves concurrent work on hardware, applications or system software.

Continued on the next page

Continued: Concurrency, Team/Customer Complexity			
Complex_Desc	Customer Complexity	• "high" • "medium" • "low"	Indicates some degree of customer complexity.
Complexity_Desc	Team Complexity	• "Single Person Single Team Multiple Teams" • "Same Location Multiple Sites Multiple Sites" • "Different Cities Multiple Sites" • "Different Time Zones Multiple Sites" • "Different Countries Subcontracted to a Third Party"	Indicates team complexity.

Table 7.3: Team/Management Experience

Variable	Descriptive Name	Values	Comments
sys_experience _Experience_Desc	System Experience	• "Less Than 1 Year" • "1 - 3 Years" • "Greater Than 3 Years"	Indicates the development team's experience with the system.
tool_experience _Experience_Desc	Tool Experience	• "Less Than 1 Year" • "1 - 3 Years" • "Greater Than 3 Years"	Indicates the development team's experience with the development tools.
info_tech _experience _Experience_Desc	Information Technology Experience	• "Less Than 1 Year" • "1 - 3 Years" • "Greater Than 3 Years"	Indicates the development team's experience with the information technology.

Continued on the next page

Continued: Team/Management Experience			
language_experience_Experience_Desc	Language Experience	• "Less Than 1 Year" • "1 - 3 Years" • "Greater Than 3 Years"	Indicates the development team's experience with the programming language.
computer_experience_Experience_Desc	Computer Experience	• "Less Than 1 Year" • "1 - 3 Years" • "Greater Than 3 Years"	Indicates the development team's experience with the computers.
methodology_experience_Experience_Desc	Methodology Experience	• "Less Than 1 Year" • "1 - 3 Years" • "Greater Than 3 Years"	Indicates the development team's experience with the adopted methodology.
Proj_Mgt_experience_Experience_Desc	Project Management Experience	• "Less Than 1 Year" • "1 - 3 Years" • "Greater Than 3 Years"	Indicates the project management's experience.

Table 7.4: General Project Attributes

Variable	Descriptive Name	Values	Comments
Proj_ID	Project ID	numerical	This is the ID with which the projects can be retrieved from the database.
Proj_Close_Date	Project Close Date	date	The date the project was finished and closed.
Proj_Full_Name	Project Name	alphanumerical	The name the project is entered with in the database.
Unadj_FP _Count_Sum	Unadjusted FP	numerical	The size of the project in FPs.
SumOfEffort _Hours	Effort	numerical	The amount of effort hours the project needed for completion.
Proj_Start_Date	Project Start Date	date	The date the project was started.

Table 7.5: Country, Industry Sector, Project Type

Variable	Descriptive Name	Values	Comments
Country_Name	Country	character	The country the project was developed in.
Industry_Desc	Industry Sector	character	The industry sector the project was developed for.
Proj_Type_Desc	Project Type	character	The type of project.

Table 7.6: Derived Variables

Variable	Descriptive Name	Values	Comments
Delivery	Delivery Rate	numerical	Derived delivery rate from the original data (effort divided by size).
Productivity	Productivity	numerical	Derived productivity from the original data(size divided by effort).

Appendix D

Results from Investigation Based on Simulated Data

Table 7.7: SDS1 Robust Filtering Classifications

	True Noise Y	True Noise N
Robust Filtering Noise Y	0	6
Robust Filtering Noise N	12	105

Table 7.8: SDS1 PF 1+ Classifications

	True Noise Y	True Noise N
PF 1+ Noise Y	10	49
PF 1+ Noise N	2	62

Table 7.9: SDS1 PF 2+ Classifications

	True Noise Y	True Noise N
PF 2+ Noise Y	5	8
PF 2+ Noise N	7	103

Table 7.10: SDS2 Robust Filtering Classifications

	True Noise Y	True Noise N
Robust Filtering Noise Y	5	8
Robust Filtering Noise N	32	78

Table 7.11: SDS2 PF 1+ Classifications

	True Noise Y	True Noise N
PF 1+ Noise Y	21	21
PF 1+ Noise N	16	65

Table 7.12: SDS2 PF 2+ Classifications

	True Noise Y	True Noise N
PF 2+ Noise Y	10	0
PF 2+ Noise N	27	86

Table 7.13: SDS3 Robust Filtering Classifications

	True Noise Y	True Noise N
Robust Filtering Noise Y	19	9
Robust Filtering Noise N	55	40

Table 7.14: SDS3 PF 1+ Classifications

	True Noise Y	True Noise N
PF 1+ Noise Y	44	14
PF 1+ Noise N	30	35

Table 7.15: SDS3 PF 2+ Classifications

	True Noise Y	True Noise N
PF 2+ Noise Y	22	1
PF 2+ Noise N	52	48

Table 7.16: SDS4 Robust Filtering Classifications

	True Noise Y	True Noise N
Robust Filtering Noise Y	35	4
Robust Filtering Noise N	76	8

Table 7.17: SDS4 PF 1+ Classifications

	True Noise Y	True Noise N
PF 1+ Noise Y	73	4
PF 1+ Noise N	38	8

Table 7.18: SDS4 PF 2+ Classifications

	True Noise Y	True Noise N
PF 2+ Noise Y	41	0
PF 2+ Noise N	70	12

Table 7.19: SDS1 Robust Filtering Mean Comparison

Data set/Technique	Residuals Comparison	
	Do Nothing	Robust Filtering
SDS1 Robust Filtering	Sample Size = 123	Sample Size = 117
	Mean = 89.876	Mean = 94.485
	Rank Sum = 14785.5	Rank Sum = 14134.5
	Test Statistics = 7159.500	Test Statistics = 7231.500
	Expectation of Test Statistics = 7195.500 Variance of Test Statistics = 289019.250 Z-Score 0.067 One-Sided P-Value for Do Nothing<Robust Filtering: 0.473 Two-Sided P-Value for Do Nothing not equal to Robust Filtering: 0.947	

Table 7.20: SDS1 Predictive Filtering Mean Comparison

Data set/Technique	Residuals Comparison
SDS1 Predictive Filtering	Do Nothing: Sample Size = 123; Mean = 89.876; Rank Sum = 11946.0; Test Statistics = 4320.000 Predictive Filtering: Sample Size = 64; Mean = 0.000; Rank Sum = 5632.0; Test Statistics = 3552.000 Expectation of Test Statistics = 3936.000 Variance of Test Statistics = 123328.000 Z-Score -1.093 One-Sided P-Value for Predictive Filtering<Do Nothing: 0.137 Two-Sided P-Value for Do Nothing not equal to Predictive Filtering: 0.274

Table 7.21: SDS1 Filtering and Polish Mean Comparison

Data set/Technique	Residuals Comparison
SDS1 Filtering and Polish	Do Nothing: Sample Size = 123; Mean = 89.876; Rank Sum = 12314.0; Test Statistics = 4688.000 Filtering and Polish: Sample Size = 123; Mean = 439.213; Rank Sum = 18067.0; Test Statistics = 10441.000 Expectation of Test Statistics = 7564.500 Variance of Test Statistics = 311405.250 Z-Score 5.155 One-Sided P-Value for Do Nothing<Filtering and Polish: 0.000 Two-Sided P-Value for Do Nothing not equal to Filtering and Polish: 0.000

Table 7.22: SDS2 Robust Filtering Mean Comparison

Data set/Technique	Residuals Comparison
SDS2 Robust Filtering	Do Nothing: Sample Size = 123; Mean = 234.083; Rank Sum = 14449.0; Test Statistics = 6823.000 Robust Filtering: Sample Size = 110; Mean = 232.335; Rank Sum = 12812.0; Test Statistics = 6707.000 Expectation of Test Statistics = 6765.000 Variance of Test Statistics = 263835.000 Z-Score -0.113 One-Sided P-Value for Robust Filtering <Do Nothing: 0.455 Two-Sided P-Value for Do Nothing not equal to Robust Filtering: 0.910

Table 7.23: SDS2 Predictive Filtering Mean Comparison

Data set/Technique	Residuals Comparison
SDS2 Predictive Filtering	Do Nothing: Sample Size = 123; Mean = 234.083; Rank Sum = 13225.0; Test Statistics = 5599.000 Predictive Filtering: Sample Size = 81; Mean = 86.679; Rank Sum = 7685.0; Test Statistics = 4364.000 Expectation of Test Statistics = 4981.500 Variance of Test Statistics = 170201.250 Z-Score -1.497 One-Sided P-Value for Predictive Filtering<Do Nothing: 0.067 Two-Sided P-Value for Do Nothing not equal to Predictive Filtering: 0.134

Inner comparison:

Do Nothing	Predictive Filtering
Sample Size = 123	Sample Size = 81
Mean = 234.083	Mean = 86.679
Rank Sum = 13225.0	Rank Sum = 7685.0
Test Statistics = 5599.000	Test Statistics = 4364.000

Table 7.24: SDS2 Filtering and Polish Mean Comparison

Data set/Technique	Residuals Comparison	
	Do Nothing	Filtering and Polish
SDS2 Filtering and Polish	Sample Size = 123	Sample Size = 123
	Mean = 234.083	Mean = 618.472
	Rank Sum = 13874.0	Rank Sum = 16507.0
	Test Statistics = 6248.000	Test Statistics = 8881.000
	Expectation of Test Statistics = 7564.500	
	Variance of Test Statistics = 311405.250	
	Z-Score 2.359	
	One-Sided P-Value for Do Nothing <Filtering and Polish: 0.009	
	Two-Sided P-Value for Do Nothing not equal to Filtering and Polish: 0.018	

Table 7.25: SDS3 Robust Filtering Mean Comparison

Data set/Technique	Residuals Comparison
SDS3 Robust Filtering	Do Nothing: Sample Size = 123; Mean = 506.421; Rank Sum = 13589.5; Test Statistics = 5963.500 Robust Filtering: Sample Size = 95; Mean = 499.704; Rank Sum = 10281.5; Test Statistics = 5721.500 Expectation of Test Statistics = 5842.500 Variance of Test Statistics = 213251.250 Z-Score -0.262 One-Sided P-Value for Robust Filtering<Do Nothing: 0.397 Two-Sided P-Value for Do Nothing not equal to Robust Filtering: 0.793

Do Nothing	Robust Filtering
Sample Size = 123	Sample Size = 95
Mean = 506.421	Mean = 499.704
Rank Sum = 13589.5	Rank Sum = 10281.5
Test Statistics = 5963.500	Test Statistics = 5721.500

Table 7.26: SDS3 Predictive Filtering Mean Comparison

Data set/Technique	Residuals Comparison
SDS3 Predictive Filtering	Do Nothing: Sample Size = 123; Mean = 506.421; Rank Sum = 12351.5; Test Statistics = 4725.500 Predictive Filtering: Sample Size = 65; Mean = 331.344; Rank Sum = 5414.5; Test Statistics = 3269.500 Expectation of Test Statistics = 3997.500 Variance of Test Statistics = 125921.250 Z-Score -2.052 One-Sided P-Value for Predictive Filtering<Do Nothing: 0.020 Two-Sided P-Value for Do Nothing not equal to Predictive Filtering: 0.040

Do Nothing	Predictive Filtering
Sample Size = 123	Sample Size = 65
Mean = 506.421	Mean = 331.344
Rank Sum = 12351.5	Rank Sum = 5414.5
Test Statistics = 4725.500	Test Statistics = 3269.500

Table 7.27: SDS3 Filtering and Polish Mean Comparison

Data set/Technique	Residuals Comparison
SDS3 Filtering and Polish	**Do Nothing:** Sample Size = 123; Mean = 506.421; Rank Sum = 14219.5; Test Statistics = 6593.500 **Filtering and Polish:** Sample Size = 123; Mean = 721.583; Rank Sum = 16161.5; Test Statistics = 8535.500 Expectation of Test Statistics = 7564.500 Variance of Test Statistics = 311405.250 Z-Score 1.740 One-Sided P-Value for Do Nothing<Filtering and Polish: 0.041 Two-Sided P-Value for Do Nothing not equal to Filtering and Polish: 0.082

Table 7.28: SDS4 Robust Filtering Mean Comparison

Data set/Technique	Residuals Comparison	
SDS4 Robust Filtering	Do Nothing	Robust Filtering
	Sample Size = 123	Sample Size = 84
	Mean = 649.489	Mean = 733.015
	Rank Sum = 12466.0	Rank Sum = 9062.0
	Test Statistics = 4840.000	Test Statistics = 5492.000
	Expectation of Test Statistics = 5166.000	
	Variance of Test Statistics = 179088.000	
	Z-Score 0.770	
	One-Sided P-Value for Do Nothing<Robust Filtering: 0.221	
	Two-Sided P-Value for Do Nothing not equal to Robust Filtering: 0.441	

Table 7.29: SDS4 Predictive Filtering Mean Comparison

Data set/Technique	Residuals Comparison
SDS4 Predictive Filtering	Do Nothing: Sample Size = 123; Mean = 649.489; Rank Sum = 11104.0; Test Statistics = 3478.000 Predictive Filtering: Sample Size = 46; Mean = 432.308; Rank Sum = 3261.0; Test Statistics = 2180.000 Expectation of Test Statistics = 2829.000 Variance of Test Statistics = 80155.000 Z-Score -2.292 One-Sided P-Value for Predictive Filtering<Do Nothing: 0.011 Two-Sided P-Value for Do Nothing not equal to Predictive Filtering: 0.022

Do Nothing	Predictive Filtering
Sample Size = 123	Sample Size = 46
Mean = 649.489	Mean = 432.308
Rank Sum = 11104.0	Rank Sum = 3261.0
Test Statistics = 3478.000	Test Statistics = 2180.000

Table 7.30: SDS4 Filtering and Polish Mean Comparison

Data set/Technique	Residuals Comparison
SDS4 Filtering and Polish	Do Nothing: Sample Size = 123; Mean = 649.489; Rank Sum = 14218.5; Test Statistics = 6592.500 Filtering and Polish: Sample Size = 123; Mean = 813.670; Rank Sum = 16162.5; Test Statistics = 8536.500 Expectation of Test Statistics = 7564.500 Variance of Test Statistics = 311405.250 Z-Score 1.742 One-Sided P-Value for Do Nothing<Filtering and Polish: 0.041 Two-Sided P-Value for Do Nothing not equal to Filtering and Polish: 0.082

Inner table:

Do Nothing	Filtering and Polish
Sample Size = 123	Sample Size = 123
Mean = 649.489	Mean = 813.670
Rank Sum = 14218.5	Rank Sum = 16162.5
Test Statistics = 6592.500	Test Statistics = 8536.500