An Algorithmic Approach to Missing Data Problem in Modeling Human Aspects in Software Development

Conference or Workshop Item

An Algorithmic Approach to Missing Data Problem in Modeling Human Aspects in Software Development

Gul Calikli
Data Science Laboratory
Dept. of Mechanical and Industrial Engineering
Ryerson University
gcalikli@ryerson.ca

Ayse Bener
Data Science Laboratory
Dept. of Mechanical and Industrial Engineering
Ryerson University
ayse.bener@ryerson.ca

ABSTRACT

Background: In our previous research, we built defect prediction models by using confirmation bias metrics. Due to confirmation bias developers tend to perform unit tests to make their programs run rather than breaking their code. This, in turn, leads to an increase in defect density. The performance of prediction model that is built using confirmation bias was as good as the models that were built with static code or churn metrics.
Aims: Collection of confirmation bias metrics may result in partially "missing data" due to developers' tight schedules, evaluation apprehension and lack of motivation as well as staff turnover. In this paper, we employ Expectation-Maximization (EM) algorithm to impute missing confirmation bias data.
Method: We used four datasets from two large-scale companies. For each dataset, we generated all possible missing data configurations and then employed Roweis' EM algorithm to impute missing data. We built defect prediction models using the imputed data. We compared the performances of our proposed models with the ones that used complete data.
Results: In all datasets, when missing data percentage is less than or equal to 50% on average, our proposed model that used imputed data yielded performance results that are comparable with the performance results of the models that used complete data.
Conclusions: We may encounter the "missing data" problem in building defect prediction models. Our results in this study showed that instead of discarding missing or noisy data, in our case confirmation bias metrics, we can use effective techniques such as EM based imputation to overcome this problem.

Categories and Subject Descriptors

D.4.8 [**Software Engineering**]: Performance—*modeling and prediction*; D.m [**Software Engineering**]: Miscellaneous—*software psychology*

PROMISE '13, October 9, 2013, Baltimore, USA

General Terms

Algorithms, Human Factors, Measurement

Keywords

Software defect prediction, handling missing data, Expectation Maximisation (EM) algorithm, confirmation bias

1. INTRODUCTION

Along with many researchers in the community [16, 21], we have been building models, which employ data mining and Artificial Intelligence (AI) techniques to guide software developers, testers and project managers in decision making under uncertainty [22, 34, 35]. Such models use mostly *product* and *process* attributes of software to uncover certain patterns in software development process. The ultimate goal is to better allocate resources, to find more defects, improve product and process quality, to better estimate cost and duration of the project, to prioritize release decisions, and so forth.

On the other hand, besides *product* and *process* attributes, *people* aspect is also among the three pillars of software development (3Ps). Software development is a domain that is very much people dependent. Calling it an engineering field, and making it process driven does not change this fact much. Therefore, we need to better understand people in this process as individuals. In the literature there are prediction models, which employ people related aspects, such as organizational complexity [23], developer experience [40], and social interaction among developers [3]. However, among people related aspects, the thought processes and problem solving skills of people also have a significant impact on software quality [29, 32]. While solving problems in daily life, people use heuristics to solve problems. When heuristics fail to produce a correct judgment, it results in cognitive biases. Cognitive biases, which are defined as patterned deviations of human thought from the laws of logic and mathematics, are a likely cause of software defects [29].

As a starting point to better understand people as individuals, we have so far focused on a specific type of cognitive bias namely *confirmation bias* [5, 6, 7, 8]. Confirmation bias is defined as the tendency of people to seek evidence that verifies a hypothesis rather than seeking evidence to falsify a hypothesis and it might show up during daily software engineering activities [29]. In our recent research, we measured the confirmation biases of developers and testers in four different organizations in Canada and Turkey [5, 6, 7, 8]. For this purpose, we conducted written and interac-

tive tests with 227 subjects consisting of a pilot group of 28 people and 199 software engineers (i.e. developers, testers, analysts and project managers). As a result of the empirical analysis based on the outcomes of these tests, we found a direct correlation between confirmation biases of developers and the defect proneness of the code [7]. In our recent research, we also used confirmation bias metrics to build a defect prediction model and by using only confirmation bias metrics, we obtained defect prediction results, which are comparable with the results obtained by using static code and churn metrics [7]. Confirmation bias is just one aspect of cognitive biases, and biases are just tiny aspect of human cognition. Therefore, our results encourage us to deepen our understanding of people's cognition to better understand the blind spots in software development and to use people's cognitive metrics to build various prediction models.

Data collection in software development organizations has always been a problem [17, 35]. Collecting data through interviews and tests from the development team may also be quite challenging. As a result, during field studies involving people (i.e., software engineers), we may encounter "missing data" problem due to one or more of the following reasons:

- **Tight Schedules:** In many cases, developers have tight schedules to rush the code for the new release and therefore they may see the data collection process as waste of time.

- **Evaluation Apprehension:** Evaluation apprehension is a common problem and it is a threat to construct validity [36]. Many people are anxious about being evaluated, and some people are even phobic about testing and measurement situations. Due to evaluation apprehension, tests may not reflect actual performance subjects leading to noisy data. Moreover, people may not be willing about taking tests due to their anxiety about being evaluated, in which case we encounter the "missing data" problem.

- **Staff Turnover:** Some of the reused components of the software may have been coded by developers who left the company.

- **Lack of Motivation:** Developers may not see the direct benefit of participating such filed studies in their daily work. [17].

One way of tackling "missing data" problem is to use Artificial Intelligence (AI) techniques. There are various AI techniques that were used to handle missing data problem in other domains such as computer vision, robotics, health care, entertainment, etc. to build models/recommendation systems [2, 4, 19, 24, 33]. In empirical software engineering literature, some methods have been employed to handle missing data problem in especially cost estimation models [28, 30, 37]. In this paper, instead of ignoring incomplete data while building defect prediction models, we handle the "missing data" problem by employing Roweis' Expectation-Maximisation [26] algorithm.

The rest of the paper is organized as follows: In section II we talk about confirmation bias as a human aspect in software development and the methodology to quantify confirmation bias. Section III discusses missing data problem in building prediction models. Section IV explains our proposed algorithm to handle the missing data problem. Details of our empirical work are given in Section V. We present the results of our empirical work in Section VI and address threats to validity in Section VII. Finally, we conclude and mention future research directions in Section VIII.

2. CONFIRMATION BIAS AS A HUMAN ASPECT IN SOFTWARE DEVELOPMENT

In cognitive psychology, confirmation bias is defined as the tendency of people to seek for evidence that could verify their hypotheses rather than seeking for evidence that could falsify them. The term confirmation bias was first used by Peter Wason in his rule discovery experiment [38] and later in his selection task experiment [39].

In Wason's Rule Discovery Task, subjects are asked to discover a simple rule about triples of numbers [38]. The experimental procedure can be explained as follows: Initially, the subject is given a record sheet on which the triple "2 4 6" is written and (s)he is told that "2 4 6" conforms to this rule. In order to discover the rule, the subject is asked to write down triples together with the reasons of his/her choice on the record sheet. After each instance, the examiner tells the subject whether the instance conforms to the rule or not. The subject can announce the rule only when (s)he is highly confident. If the subject fails to discover the rule at the first attempt, (s)he can continue giving instances together with reasons for his/her choice. This procedure continues iteratively until either the subject discovers the rule or (s)he wishes to give up. However, if the subject cannot discover the rule in 45 min, the experimenter aborts the procedure. Wason designed this experiment in such a way that subjects mostly shows a tendency to focus on a set of triples that were most specific than the correct rule. Consequently, the discovery of the correct rule is possible only by following a hypothesis testing strategy, which includes tendency to refute hypotheses [25].

In Wason's Selection Task, the subject is given four cards, where each card has a letter on one side and a number on the other side. These four cards are placed on a table showing D, K, 3, 7, respectively. Given the rule "Every card that has a D on one side has a 3 on the other side", the subject is asked which card(s) must be turned over to find out whether the rule is true or false [39].

2.1 Confirmation Bias in Relation to Software Development

Due to confirmation bias, developers may perform only the unit tests that make their program work rather than breaking the code. This may lead to an increase in software defect density. Ideally, during all levels of software testing, including unit testing, a systematic hypothesis testing procedure should be followed similar to the one followed by a scientist performing experiments in his/her laboratory. In general, scientific inferences are based on the principle of eliminating hypotheses while provisionally accepting the remaining ones. Therefore, similar to a scientist, a software developer should try test scenarios starting from the ones that are less likely to fail the code and then move to test scenarios that aim for the code to fail. In most cases, there are an infinite number of scenarios that require following a strategy to select the appropriate tests. Therefore, within

the context of software development and testing, we extend the definition of confirmation bias to include one or both of the following: (1) the tendency to verify software code and (2) the incompetency to apply strategies to try to fail software code.

2.1.1 Wason's Rule Discovery Task in Relation to Developer Performance

There are similarities between Wason's rule discovery task and functional (black-box) testing that are performed by software developers to test the functional units of their codes during unit testing. This similarity is also mentioned by Teasley et al. [32]. According to the findings of Wason's rule discovery task, the subjects have a tendency to select many triples (i.e., test cases) that are consistent with their hypotheses and few tests that are inconsistent with them. Similarly, program testers may select many test cases consistent with the program specifications (positive tests) and a few that are inconsistent with them (negative tests). Moreover, the number of possible test cases is either infinite or too large to be tested within a limited amount of time. Consequently, a strategic approach must be followed that covers both positive and negative test cases while trying to make the code fail during testing in order to find as many defects as possible.

2.1.2 Wason's Selection Task in Relation to Developer Performance

Wason's selection task measures the capability of the subject to use logical rules such as modus ponens and modus tollens as well as his/her tendency to refute a given statement. In unit testing when covering possible scenarios, logical reasoning is required. Moreover, testing the correctness of conditional statements in the source code during white box testing also requires logical reasoning skills. In order to explain the analogy between Wason's selection task and white box testing, we extend the example given by Stacy and MacMillian [29] as follows: Suppose a developer wants to make sure that every instance of a class named "Controller" has been initialized throughout his/her code. In unit testing, the developer would perform a test that could be thought of as checking the validity of the following hypothesis: "If an instance's class is Controller, then it has been initialized." In that case, we can categorize parts of the code that may need to be tested as follows:

- category #1: The parts of the code with instances of Controller that may or may not be initialized.
- category #2: The parts of the code with instances of a class other than Controller that may or may not be initialized.
- category #3: The parts of the code with initialized instances whose class is unknown.
- category #4: The parts of the code with uninitialized instances whose class is unknown.

Logical expression for the hypothesis "If an instance's class is Controller, then it has been initialized" would be "if p, then q", where p stands for the phrase "an instance's class is Controller" and q stands for the phrase "it (class) has been initialized". According to modus ponens, given that p is true, "if p then q" is true only if q is true. Therefore, one must check all instances of the class "Controller" to guarantee that they have all been initialized. This means that the parts of the code that fall into category #1 must be tested. However, this is not adequate. Since "if p, then q" is equivalent to "if not-q, then not-p", one must also check the validity of the negated form of the hypothesis, which is "If an instance has not been initialized, then the class of that instance is not Controller". According to modus tollens, given that not-q is true, not-p must also be true. This means that every instance that has not been initialized must be checked to find out whether the class of that instance is "Controller" or not. Therefore, a developer must also test the parts of the code that fall into category #4.

2.2 Methodology to Quantify Confirmation Bias

In order to perform empirical analysis, it was necessary to quantify/measure confirmation biases of software engineers. For this purpose, we developed a methodology to define a confirmation bias metrics set and to extract metrics values. Details of our methodology to define and extract confirmation bias metrics can be found in our previous work [7] and it mainly consists of the following steps:

2.2.1 Preparation of the Confirmation Bias Test

Confirmation bias test consists of written questions and an interactive question. Interactive question is Wason's Rule Discovery Task itself [39], while written test is based on Wason's Selection Task [38]. Written test consists of two parts: the first part consists of 7 abstract and 7 thematic questions. Abstract questions requires logical reasoning skills to be answered correctly, while real life experience and/or memory cueing [12] can help to answer thematic questions correctly. Both abstract and thematic questions were prepared based on the experiments conducted in cognitive psychology literature experiments to show the exitance of confirmation bias among people [12, 38, 39]. The second part of the test consists of 9 thematic questions with software development and testing theme. Initially, we administered confirmation test to a pilot group consisting of 28 Computer Engineering PhD candidates. Half of the participants in the pilot group had at least two years of commercial software product development experience. After pilot study, we administered the test to software engineers in various large scale companies and Small Medium Enterprises (SMEs). In addition to a pilot group consisting of 28 subjects, so far we have administered the confirmation bias test to 199 software engineers (129 developers, 26 testers, 32 analysts and 12 project managers) from 4 large scale companies and 3 SMEs.

2.2.2 Formation of the Confirmation Bias Metrics Set

Some of the metrics in the metrics set were inherited from cognitive psychology literature, while the rest were defined as a result of the observations we made during the administration of the confirmation bias test. The initial metric suite was formed concurrently with the preparation of the interactive question and the set of written questions. Statistical analysis and feature selection techniques helped to eliminate metrics that displayed a lower level of significance in the measurement/quantification of confirmation bias. Our metrics set evolved as our research progressed [5, 6, 8] and took its final form in [7]. Table 1 lists the final set of metrics obtained from interactive question and the written questions.

Table 1: Confirmation Bias Metrics Set

Metric	Explanation	Test Type
N_A	Number of rule announcements	Interactive
T_I	Duration of interactive question session (in minutes)	Interactive
$Ind_{elim/enum}$	Eliminative/enumerative by Wason	Interactive
$F_{negative}$	Frequency of negative instances	Interactive
F_{IR}	Immediate rule announcement frequency	Interactive
$avgLIR$	Average length of immediate rule announcements	Interactive
$Instances/Time$	Number of instances given per unit time	Interactive
$UnqReasons/Time$	Number of unique reasons given per unit time	Interactive
$Rules/Time$	Number of rules announced per unit time	Interactive
$UnqRules/Time$	Number of unique rules that are announced per unit time	Interactive
S_{Abs}	Score in abstract questions	Written
S_{Th}	Score in thematic questions	Written
S_{SW}	Score in the second part of the written question set	Written
T_{Th-Abs}	Time it takes to answer the first part of the written question set	Written
T_{SW}	Time it takes to answer the second part of the written question set	Written
$ABS_{CompleteInsight}$	Number of abstract questions answered with complete insight	Written
$ABS_{PartialInsight}$	Number of abstract questions answered with partial insight	Written
$ABS_{NoInsight}$	Number of abstract questions answered with no insight	Written
$Th_{CompleteInsight}$	Number of thematic questions answered with complete insight	Written
$Th_{PartialInsight}$	Number of thematic questions answered with partial insight	Written
$Th_{NoInsight}$	Number of thematic questions answered with no insight	Written
$NFalsifier$	Total number of answers with only falsifying tendency	Written
$N_{Verifier}$	Total number of answers with only verifying tendency	Written
$N_{Matcher}$	Total number of answers with only matching tendency	Written
N_{None}	Total number of answers with no defined tendency	Written

3. MISSING DATA PROBLEM IN PREDICTION MODELS

Handling missing and noisy data has been a long time research problem in the field of artificial intelligence and data mining. Various imputation techniques have been used to deal with missing data while building prediction models in domains such as computer vision [4, 19], robotics [33], health care [24] and software cost/effort estimation [9, 28, 30]. Recommender systems, which are integral part of the e-commerce web sites such as Netflix and Amazon, and social web sites such as YouTube are often based on statistical models that are estimated from data sets containing a very high portion of missing ratings [2].

In the literature, there are various studies, which attempt to form a taxonomy of the techniques to handle missing data [10, 18]. According to the taxonomy proposed by Little and Rubin [18], we can categorize missing data methods as follows:

- **Procedures Based on Completely Recorded Units:** This method consists of discarding incompletely recorded parts of the data and only analyze the complete parts of the data. This method may give satisfactory results only in the case of small amount of missing data and it is very likely to lead to biases.
- **Weighting Procedures:** This method is used in the case of non-response in survey data. Such methods are not recommended except in special cases where the amount of missing information is limited [18].
- **Imputation Based Procedures:** Among these procedures, commonly used ones are "hot-deck imputation", where recorded units in the sample are used to substitute values; "mean imputation", where means from sets of recorded values are submitted; regression imputation, where the missing data are estimated by predicted value from the regression on the known part of the data. However, as indicated by Dempster [11] this kind of procedures have some pitfalls, since the imputed data have substantial biases.
- **Model-Based Procedures:** These procedures are based on defining a model for the observed data and basing inferences on the likelihood or posterior distribution under that model. Advantages of these approaches are flexibility and the avoidance of ad-hoc methods.

Another category of methods to handle missing data consists of machine learning techniques such as multi-layer perceptrons (MLP), self-organizing maps (SOM) k-nearest neighbour (kNN) [15, 31], decision trees [27] and Linear Discriminant Analysis (LDA) [1]. Moreover, there are various techniques, which are used to handle missing data in software cost estimation models such as mean imputation, list-wise deletion (LD), kNN, and Expectation Maximization (EM) algorithms. EM algorithms proved to be very powerful techniques [18, 37] leading to highest accuracy results [37].

Among the model based approaches, EM algorithm is conceptually and computationally simple. Unlike ad-hoc interpolation methods, the EM algorithm estimates the maximum likelihood of the missing data directly at each iteration [13]. EM algorithm is so closely tied to the intuitive idea of filling in missing values and iterating. In other words, the EM algorithm handles missing data problem as follows:

1) Replace missing values by estimated values, 2) estimate parameters, 3) re-estimate the missing values assuming the new parameter estimates are correct, 3) re-estimate parameters, and continue iterating until convergence.

In our case, amount of missing data is high, consisting of the entire confirmation bias metrics for each entry. Therefore, rather than methods such as discarding missing data, weighting procedures and imputation based procedures, EM algorithms are suitable, in order to handle this specific missing data problem in our hand. However, one of the main drawbacks of EM algorithms is that it can be very slow to converge with large fractions of missing data. [18]. In some cases, the M-step may not have closed form so that the theoretical simplicity of the EM Algorithm does not convert to practical simplicity. In such cases the efficiency can be increased by combining the EM algorithm with various other techniques [20]. In this research, we employ a variation of EM algorithm, which is capable of efficiently handling missing data problem with large data in high dimensions.

4. ROWEIS' EM ALGORITHM

Compared to other variations of the EM algorithm, which rely on complete-data computations, Roweis' EM algorithm allows simple and efficient computation to handle the missing data problem while dealing with large data in high dimensions (e.g. up to 2^{17} data points in 2^{12} dimensions) [26]. Roweis' algorithm also inspired Bell et al. to develop their prize winning matrix factorization algorithm to improve the accuracy of large recommender systems in the presence of high amount of missing data such as Netflix Cinematch [2]. For these all these reasons, we decided to employ Roweis' EM algortihm to deal with developers' missing confirmation bias data, while building defect prediction models.

Roweis' algorithm originally aimed to perform PCA factorization in the presence of missing data. However, the algorithm can be employed to directly handle the missing data problem, as it was done by Bell et al. [2]. In order to explain the fundamentals of Roweis' algorithm, we must refer to the goal of PCA, which is to find a mapping from the data Y in the original d-dimensional space to a new k-dimensional space where $k < d$, such that there is minimum loss of information.

$$X = w^T Y \tag{1}$$

Equation 1 is equivalent to $Y = CX$, where C is equal to $(w^T)^{-1}$. We can reformulate equation 1 as $X = C^{-1}Y = C^{-1}IY$, where $I = (C^T)^{-1}C^T$ is the identity matrix. As a result the, "Expectation" step (E-step) of Roweis' algorithm can be formulated as:

$$X = (C^T C)^{-1} C^T Y \tag{2}$$

In Equation 2, Y is a dxn matrix of the original data and X is kxn matrix of the unknown states. The columns of C will span the first k principle components of Y. The "Maximization" step (M-step) of the algorithm can be reformulated as in Equation 3:

$$C = YX^T(XX^T)^{-1} \tag{3}$$

During the E-step of the EM algorithm, for any data entry y in the data matrix Y with some of its coordinates (i.e. attributes) missing, a unique pair x^* and y^* can be calculated

```
% Initially set matrix C randomly
for all iteration in [1 : 1 : MAX(iter)] do
  % E-Step of EM Algorithm
  for all y in Y do
    for all i in [1:1:size(y,1)] do
      % Missing Data Imputation is done within
      E-step
      if isMissing(y(i)) == TRUE then
        % Find Least Squares Solution to
          Cx = y by QR decomposition
        if isSparseMatrix(C) == TRUE then
          [Q,R] = qrDecompose(C)
        else
          R = upperTriangle(qrDecompose(C))
        end if
        x = R^-1 (R^T)^-1 C^T y
        r = y - Cx
        e = R^-1 (R^T)^-1 C^T r
        x = x + e
      end if
      if isMissing(y(i)) == FALSE then
        x = (C^T C)^-1 C^T y
      end if
      X(i, :) = x;
      % Assign x to i^th row of matrix X.
    end for
  end for

  % M-Step of EM Algorithm
  C = YX^T (XX^T)^-1

end for
```

Figure 1: Pseudocode for Roweis' EM Algorithm

such that $||Cx - y||$ is minimized. In other words, missing values can be imputed by solving the least squares problem for $||Cx - y = 0||$.

5. EMPIRICAL STUDY

5.1 Datasets

In this study, we used datasets from four different projects as shown in Table 2. In order to build the defect prediction model, we took into account only the source code files whose development activities can be traced through the version control system. Only these active source code files were tested by the testing teams. Therefore, project managers needed guidance about defect-prone parts of these files to efficiently allocate their testing resources within tight release deadlines. In Table 2, the total number of maintained/developed files, file types and defect rates are listed for each dataset. The defect rate is the ratio of the number of defective files to the number of active files.

Dataset ERP belongs to a project group that consists of six developers who are employees of the largest ISV (independent software vendor) in Turkey. The software developed by this project group is an enterprise resource planning (ERP) software. The snapshot of the software that was retrieved from the version management system dates back to March 2011, and it consists of 3,199 java files. The remaining three datasets come from the largest wireless telecom oper-

Table 2: Properties of datasets.

Dataset	# of Active Files	Defect Density	# of Developers
ERP	3199	0.07	6
Telecom1	826	0.11	7
Telecom2	1481	0.03	4
Telecom4	63	0.05	10

ator (GSM) company in Turkey. Dataset Telecom1 consists of four versions of a software product that is used to launch new campaigns. On average, 545 java files exist in a single version, and they make modifications to 206 files per version (also on average). The remaining two datasets come from the billing and charging system. Among these two projects, the Telecom2 dataset is relatively a new one, and it consists of both java and JSP files. The modification and updates involve all existing source code files in the project as well as the creation of new files.

Dataset Telecom3 is extracted from the database transactions system. This software package has been developed and maintained since the inception of the GSM company in 1994 and consists of PL/SQL files. Similar to Telecom1, only the files that are maintained are taken into account in the defect prediction analysis.

5.2 Methodology

Our goal in this research is to compare the defect prediction performance of a model that is built with the complete set of confirmation bias metrics with the models that are built with incomplete set of confirmation bias metrics. Therefore, we built prediction models for the complete form of each dataset. For each dataset, we also generated partially missing datasets corresponding to each possible combination of missing developers. This resulted in the formation of $2^N - 2$ different missing data configurations. Later, we imputed each partially missing data by using Roweis' EM algorithm and used the imputed dataset to build defect prediction models.

5.2.1 Construction of the Prediction Model

In this study, we used the Naïve Bayes algorithm since it combines signals coming from different attributes and it also performs well in software defect prediction [16, 21]. As shown in Table 2, the datasets are imbalanced; the number of defective files is far less than the number of defect-free files. Therefore, we used the under-sampling method, which is the most suitable sampling method for our datasets [21]. In order to overcome ordering effects, we shuffled the data ten times, and a ten-fold cross validation was used for each ordering configuration of input data. Therefore, for each ordering configuration, we created ten stratified bins: nine of these ten bins are used as training sets, and the last one is used as the test set [14]. As a result, during each experiment, the Naïve Bayes algorithm with under-sampling is executed $10X10 = 100$ times for each dataset.

We formed the defect prediction analysis at the granularity level of "file" since defect data were not available at the granularity level of "method" in either of the two software companies. Each file is developed by a group, which consists of one or more developers. Therefore, in the input data, which is used to build the prediction models, each file is represented by the confirmation bias metrics of the corresponding developer group. We used three different operators to calculate the minimum, maximum and average values of the metrics of developers who committed code to the same source file. Assuming that A_{di} represents the i^{th} confirmation bias metric value of d^{th} developer, $d \in G_j$ means that d^{th} developer is among the group of developers who created and/or modified j^{th} source file, and and finally, S_{ji}^{op} represents the resulting i^{th} confirmation bias metric value of j^{th} source file when operator op is applied. op can be one of the operators min , max or avg which are used to find minimum, maximum and average values of the i^{th} confirmation bias metric respectively. We can formulate the definition for the min, max and avg operators as follows:

$$S_{ji}^{max} = max(A_{di} | \forall d \in G_j) \tag{4}$$

$$S_{ji}^{min} = min(A_{di} | \forall d \in G_j) \tag{5}$$

$$S_{ji}^{avg} = \sum_{d} (A_{di} | \forall d \in G_j) / \sum_{d} (1 | \forall d \in G_j) \tag{6}$$

5.2.2 Formation of Datasets with Partially Missing Data

As we have already stated, our goal in this experiment is to compare the performance of prediction models that are built with complete and partially complete data. In order to make a comparison, we assume that confirmation bias metrics of a subgroup of developers is unknown. This implies that confirmation bias metrics of any file created and/or updated by any member of this subgroup of developers is missing. 2^N is the total number of subsets of the set of developers, where N is the total number of developers. After we exclude the empty set and the complete set of developers, we obtain $2^N - 2$ missing data configurations. In other words, we exclude the following two cases: 1) Confirmation bias metrics of all developers are known, and 2)Confirmation bias metrics of none of the developers are known.

For each incomplete data set, we employ Roweis' EM Algorithm. The pseudo code, which explains the details of Roweis' EM Algorithm, is given in Figure 1. Output of Roweis' Algorithm is the imputed form of the incomplete dataset (i.e., $Y_{imputed} = CX$). We build defect prediction models using each of these imputed confirmation bias metric sets as input. This results in the formation of $2^N - 2$ defect predictors. Prediction performance of each of these predictors is compared with the performance of the prediction model that is built using the complete confirmation bias metric set. Percentage of missing data (i.e. missing %) for each incomplete data set can be calculated as follows:

$$missing\% = N_{missing} / (N_{missing} + N_{complete}) \tag{7}$$

In Equation 7, $N_{complete}$ is total number of complete entries in the data set, and $N_{missing}$ corresponds to total number of entries with missing confirmation bias metric values. Each entry in the data set corresponds to a source code file and it consists of the confirmation bias metric values of the group of developers who created and/or updated that file. Confirmation bias metric values of each developer are estimated from the outcomes of confirmation bias tests. In order to calculate confirmation bias metric values of each developer group we use, the operators defined by the Equations 4, 5 and 6. Therefore, missing confirmation bias developers, who created and/or updated a source code file, automatically implies that confirmation bias metrics for the developer group of that file are completely missing. In other words, the entry in the data set corresponding to that file is completely missing.

5.2.3 Performance Measurement Criteria

In order to evaluate the performance of the defect predictors built by using different metric suite combinations, we used the well-known performance measures which are probability of detection, false-alarm rate and balance [21].

Probability of detection (pd) measures how good a predictor is in finding defective modules, where modules can be files, methods or packages depending on the granularity level. In the ideal case, we expect a predictor to catch all defective modules/files. This implies that pd is equal to 1.

Probability of false alarms (pf) measures false alarm rates, when predictor classifies defect-free modules as defective. In the ideal case, we expect a predictor to classify none of the defect-free modules as defective. In other words, the value of pf is equal to 0.

Balance (bal) is formulated to be the Euclidean distance from the *sweet spot* ($pd = 1$ and $pf = 0$) normalized by the maximum possible distance to this spot. In practice, the ideal case where a defect predictor has high probability of detecting defective files and low probability of false alarm is very rare. Therefore, we try to balance between pd and pf values. It is desirable that predictor performance is close to the *sweet spot* as much as possible.

$$bal = 1 - \frac{\sqrt{(1-pd)^2 + (0-pf)^2}}{\sqrt{2}} \quad (8)$$

Pd and pf values are calculated using Confusion Matrix that is given in Table 3. In the confusion matrix, TP is the number of correctly classified defective modules, FP is the number of non defective modules that are classified to be defective, FN is the number of defective modules that are classified to be non-defective and finally TN is the number of correctly classified non-defective modules. Formulations for pd and pf in terms of confusion matrix values is given below:

$$pd = TP/(TP + FN) \quad (9)$$

$$pf = FP/(FP + TN) \quad (10)$$

6. EMPIRICAL STUDY RESULTS

As mentioned previously, for each dataset we prepared $2^N - 2$ missing data configurations, where N is the total number of developers, who work in the project corresponding to that dataset. For each missing data configuration of a given dataset, we built defect prediction models after handling missing data by employing Roweis' EM algorithm. We also built defect prediction models for the complete form of that dataset. The results of the corresponding missing data configurations are presented in Tables 4, 5, 6 and 7 for datasets ERP, Telecom1, Telecom2 and Telecom3, respectively. In Tables 4, 5, 6 and 7, defect prediction results of the complete form of the mentioned datasets (i.e., Missing% = 0) are also given.

Table 3: Confusion matrix *TP:True Positives, FN:False Negatives, FP:False Positives, TN:True Negatives*

	Defective (Predicted)	Non-Defective (Predicted)
Defective (Actual)	TP	FN
Non-Defective (Actual)	FP	TN

Table 4: Defect Prediction Performance Results with Missing Data for ERP Dataset

Missing %	pd	pf	balance
0%	0.91	0.31	0.74
$(0-10]\%$	0.77	0.29	0.68
$(10-20]\%$	0.74	0.29	0.67
$(20-30]\%$	0.72	0.29	0.66
$(30-40]\%$	0.70	0.31	0.65
$(40-50]\%$	0.67	0.31	0.64
$(50-60]\%$	0.62	0.16	0.68
$(60-70]\%$	0.60	0.20	0.63
$(70-80]\%$	0.64	0.28	0.58
$(80-90]\%$	0.76	0.46	0.52

As shown in Tables 4-7, we categorized prediction performance results of missing data configurations of each dataset with respect to the corresponding missing data percentage estimated by Equation 7. For categorization purposes, we formed nine missing data percentage ranges $(n_1, n_2]$. For instance, if a missing data configuration results in 15% missing data, then the corresponding defect prediction performance results belong to the missing data percentage range $(10, 20]$, since 15% is grater than 10% and less than or equal to 20%. However, there might not be any missing data configurations, which fall within a missing data percentage range $(n_1, n_2]$. In dataset Telecom1, none of the missing data configurations results in a missing data percentage, which falls within the range $80\% - 90\%$. On the other hand, none of the missing data configurations in the dataset Telecom2 fall in the ranges $(0, 10]$, $(10, 20]$, $(30, 40]$ or $(70, 80]$. This is due to only 12 missing data configurations (i.e. there are only 4 developers) in dataset Telecom2. This amount is small compared to the total number of missing data configurations in the rest of the datasets.

As shown in Table 4 imputed form of the dataset ERP yields lower pd values compared to the pd values obtained for dataset's complete form. On the other hand, a decrease in the false positives (i.e. pf values) is observed especially for missing data percentage values that are greater then 50%. A similar trend is observed also in the dataset Telecom1, as shown in Table 5. The reason why pf values fall as miss-

Table 5: Defect Prediction Performance Results with Missing Data for Telecom1 Dataset

Missing %	pd	pf	balance
0%	0.66	0.38	0.62
$(0-10]\%$	0.65	0.33	0.64
$(10-20]\%$	0.62	0.32	0.63
$(20-30]\%$	0.60	0.31	0.63
$(30-40]\%$	0.58	0.31	0.61
$(40-50]\%$	0.56	0.31	0.58
$(50-60]\%$	0.50	0.28	0.54
$(60-70]\%$	0.46	0.26	0.51
$(70-80]\%$	0.40	0.22	0.48
$(80-90]\%$	–	–	–

ing data percentage increases can be explained by the uneven distribution of the work load among developers in both projects ERP and Telecom1. In the project group of dataset Telecom1, there were two developers, who contributed 79% and 87% of the source code, respectively (i.e. top developers). Moreover, Telecom 1 project members launch a new release of their software every ten days on average. Taking all these factors into account, especially these two top developers had a significant amount of workload, while we were administering the confirmation bias tests. We had to reschedule test dates of these developers 3 and 4 times respectively. Finally, we had the opportunity to administer confirmation bias test to these top developers, they were mentally tired and reluctant to take the test. Especially, during the interactive test, one of these developers announced the same rule by whether exactly repeating the rule itself or re-formulating/rewording it for 45 minutes. The other developer terminated the interactive test after having tried for 15 minutes. As a result, the tests did not reflect the actual performance results of these two developers. Hence, as the missing percentage introduced in the data increases, the missing confirmation bias metrics belonging to these two developers are imputed. These imputed values are closer to developers' actual performance, which they would have exhibited, if they had taken the test under normal circumstances. As a result of being mentally exhausted, noise was introduced to the confirmation bias data. Our results support that using imputation techniques can also be useful in order to remove noise in the data.

As it can be seen from the Table 6, the prediction performance results obtained for missing data percentages, which are in the ranges $(20-30]\%$ and $(40-50]\%$ are comparable with the results obtained for complete data.

In the results of Dataset Telecom3, we observe decrease in pd and an increase in pf, which is in line with our expectations as information content degrades in the presence of missing data. Unlike dataset Telecom1, there is an even distribution in terms of work loads among developers of Telecom3 software project. Moreover, duration between two consequent releases of this software project is six months, which is considerably long compared to that of the Telecom1 project, which is only 10 days. Therefore, the developers of Telecom3 project had enough time to take the confirmation bias test. In addition to this, during the administration of the confirmation bias tests, we also observed that these developers were highly motivated to take the test.

Table 6: Defect Prediction Performance Results with Missing Data for Telecom2 Dataset

Missing %	pd	pf	balance
0%	0.60	0.35	0.61
$(0-10]\%$	–	–	–
$(10-20]\%$	–	–	–
$(20-30]\%$	0.69	0.47	0.57
$(30-40]\%$	–	–	–
$(40-50]\%$	0.60	0.40	0.57
$(50-60]\%$	0.66	0.52	0.52
$(60-70]\%$	0.65	0.50	0.51
$(70-80]\%$	–	–	–
$(80-90]\%$	0.53	0.42	0.43

Table 7: Defect Prediction Performance Results with Missing Data for Telecom3 Dataset

Missing %	pd	pf	balance
0%	0.93	0.15	0.85
$(0-10]\%$	0.94	0.21	0.78
$(10-20]\%$	0.93	0.22	0.77
$(20-30]\%$	0.92	0.27	0.72
$(30-40]\%$	0.91	0.32	0.67
$(40-50]\%$	0.88	0.25	0.72
$(50-60]\%$	0.93	0.27	0.72
$(60-70]\%$	0.89	0.28	0.70
$(70-80]\%$	0.89	0.34	0.64
$(80-90]\%$	0.89	0.38	0.60

7. THREATS TO VALIDITY

Methodology for the definition and extraction of confirmation bias metrics was defined in our previous research. Therefore, threats to validity regarding the definition and extraction of confirmation bias metrics was defined in our previous paper [7]. In this section, we explain the three major threats to the validity of our experiments: construct, internal and external. To avoid the construct validity threats in relation to measurement artifacts, we used three popular performance measures in software defect prediction research: the probability of detection (pd), the probability of false positives (pf) and balance values (bal). In order to avoid internal validity threats, we shuffled data ten times and used ten-fold cross validation for each ordering configuration of the input data to overcome ordering effects. Moreover, during under-sampling we shuffled each portion of the dataset ten times (which was used as an input to the Naïve Bayes algorithm). As a result, the Naïve Bayes algorithm with under-sampling was executed 100 times for each dataset during each experiment. In order to externally validate our results, we used datasets from four different developer groups, three of which were from a telecommunication company and one from an ISV specialized in the ERP domain. Hence, our datasets cover two different software development domains. We were also able to collect datasets from two different project groups within the telecommunications company. One project group developed software that is responsible for launching GSM tariff campaigns to its customers and mainly consists of user interfaces (Dataset Telecom1). The remaining two projects (Datasets Telecom2 and Telecom3) come from the billing and charging system pri-

marily comprised of database transactions, and there is no direct interaction with the customer via user interfaces.

8. CONCLUSIONS AND FUTURE WORK

In empirical software engineering, it is crucial to understand how people make decisions and solve problems throughout the software development process. However, we mostly encounter the "missing data" problem during empirical studies, which involve people. In this paper, we proposed a solution to deal with the "missing data" problem.

In our recent research, we have particulary focused on people's decision making process, the cognition. Human cognition is a complex process to understand and model. We took into account only one trait of human cognition called confirmation bias. The reason why we picked confirmation bias is two folds: Firstly, it is one of the most researched topics in cognitive psychology including the grounded work by Wason and its extensive variations over the last sixty years [38, 39]. Secondly, there is empirical evidence showing the existence of confirmation bias among developers/testers [32]. In our recent research, we used confirmation bias metrics to build defect prediction models and by using only confirmation bias metrics, we obtained defect prediction results, which are comparable with the results obtained by using static code and churn metrics [7].

While building models/tools to guide software professionals in decision making, in most cases data is partially available or noisy. We argue that we need to find mechanisms to deal with this problem effectively, if we would like to help software engineers to run their operations more effectively and efficiently. In this paper, we employed an EM algorithm to impute missing confirmation bias metrics. Our empirical results showed that by using EM algorithm to impute missing confirmation bias metrics values and then by using the imputed data to build defect prediction models, we can achieve prediction results that are comparable with the results obtained by using complete data (i.e., confirmation bias metrics values).

Our future direction will be to include other cognitive bias types as well as to investigate different techniques to handle the "missing data" problem.

9. ACKNOWLEDGMENTS

The authors would like to thank Turgay Aytac and Ayhan Inal from Logo Business Solutions as well as to Turkcell A.S. This research is supported in part by NSERC Discovery Grant No: 402003-2012.

10. REFERENCES

[1] E. Acuna and C. Rodriguez. The treatment of missing values and its effect in the classifier accuracy. In *Classification, Clustering and Data Mining Applications*, pages 639–648, 2004.

[2] R. Bell, Y. Koren, and C. Volinsky. Modeling relationships at multiple scales to improve accuracy of large recommender systems. In *Proceedings of the 13th International Conference on Knowledge Discovery and Data Mining*, pages 95–104. ACM SIGKDD, 2007.

[3] C. Bird, N. Nagappan, H. Gall, B. Murphy, and P. Devanbu. Putting it all together: Using socio-technical networks to predict failures. In *Proceedings of the 17th International Symposium on Software Reliability Engineering*, Raleigh, NC, Nov. 2009.

[4] A. M. Buchanan and A. W. Fitzgibbon. Damped newton algorithms for matrix factorization with missing data. In *Proceedings of the 2005 Conference on Computer Vision and Pattern Recognition*, pages 187–190. IEEE Computer Society, 2005.

[5] G. Calikli, B. Arslan, and A. Bener. Confirmation bias in software development and testing: An analysis of the effects of company size, experience and reasoning skills. In *Proceedings of the 22nd Annual Psychology of Programming Interest Group Workshop (PPIG '10)*, Leganes, Spain, September 2010b.

[6] G. Calikli and A. Bener. Empirical analyses factors affecting confirmation bias and the effects of confirmation bias on software developer/tester performance. In *Proceedings of the 5th International Workshop on Predictor Models in Software Engineering*, New York, NY, USA, 2010.

[7] G. Calikli and A. Bener. Influence of confirmation biases of developers on software quality: An empirical study. *Software Quality Journal*, 21(2):377–416, March 2013.

[8] G. Calikli, A. Bener, and B. Arslan. An analysis of the effects of company culture, education and experience on confirmation bias levels of software developers and testers. In *Proceedings of the 32nd International Conference on Software Engineering (ICSE '10)*, pages 187–190, New York, NY, USA, November 2010a.

[9] M. H. Cartwright. Dealing with missing software project data. In *Proceedings of the 9th International Software Metrics Symposium*, pages 154–165, September 2003.

[10] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[11] A. P. Dempster, N. M. Laird, and D. B. Rubin. Introduction. In *in Incomplete Data in Sample Surveys (Volume 2): Theory and Bibliography*, pages 3–10. New York: Academic Press, 1983.

[12] J. S. B. T. Evans, S. E. Newstead, and R. M. Byrne. *Human Reasoning: The Psychology of Deduction*. Lawrence Erlbaum Associates Ltd, East Sussex, UK, 1993.

[13] Z. Ghahramani and M. I. Jordan. Supervised learning from incomplete data via an em approach. In *Advances in Neural Information Processing Systems*, pages 120–127. Morgan Kaufmann, 1994.

[14] M. A. Hall and G. Holmes. Benchmarking attribute selection for discrete class data mining. *IEEE Transactions on Knowledge and Data Engineering*, 15:1437–1447, 2003.

[15] J. M. Jerez, I. Molina, P. J. Garcia-Laencina, E. Alba, N. Ribelles, M. Martin, and L. Franco. Missing data imputation using statistical and machine learning methods in a real breast cancer problem. *Journal of Artificial Intelligence in Medicine*, 50(2):105–115, July 2010.

[16] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings.

IEEE Transactions on Software Engineering, 34(4):485–496, July 2008.

[17] T. C. Lethbridge, S. E. Sim, and J. Singer. Studying software engineers: Data collection techniques for software field studies. *Journal of Empirical Software Engineering*, 10:311âĂŞ341, March 2005.

[18] R. J. A. Little and D. B. Rubin. *Statistical Analysis with Missing Data*. John Wiley and Sons, Hoboken, New Jersey, 2002.

[19] M. Marques and J. Costeira. Estimating 3d from degegenerate sequences with missing data. *Journal of Computer Vision and image Understanding*, 113(2):261–272, February 2009.

[20] G. J. McLachlan and T. Krishnan. *The EM Algorithm and Extensions*. John Wiley and Sons, New York, USA, 1997.

[21] T. Z. Menzies, C. J. Hihn, and K. Lum. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 33(1):2–13, Jan. 2007.

[22] A. Misirli-Tosun, B. Caglayan, A. Mirasky, A. Bener, and N. Ruffolo. Different strokes for different folks: A case study on software metrics for different defect categories. In *Proceedings of the 2nd Workshop on Emerging Trends in Software Metrics*, pages 45–51, Waikiki, Honolulu, HI, May 2011.

[23] N. Nagappan, B. Murphy, and V. R. Basili. The influence of organizational structure on software quality: An empirical case study. In *Proceedings of the 30th International Conference on Software Engineering*, pages 521–530, Lepizig, Germany, May 2008.

[24] C. M. Norrisa, W. A. Ghalia, M. L. Knudtsona, C. Naylora, and L. Saundersa. Dealing with missing data in observational health care outcome analyses. *Journal of Clinical Epidemiology*, 53(4):377–383, April 2000.

[25] F. Poletiek. *Hypothesis-testing behaviour*. Psychology Press, East Sussex, UK, 2001.

[26] S. Roweis. Em algorithms for pca and spca. In *in Advances in Neural Information Processing Systems*, pages 626–632. MIT Press, 1998.

[27] M. Saar-tsechansky, F. Provost, and R. Caruana. Handling missing values when applying classification models. *Journal of Machine Learning Research. Forthcoming*.

[28] P. Sentas and L. Angelis. Categorical missing data imputation for software cost estimation by multinomial logistic regression. *Journal of Systems and Software*, 79(3):404–414, March 2006.

[29] W. Stacy and J. Macmillan. Cognitive bias in software engineering. *Communication of the ACM*, 38(6):57–63, June 1995.

[30] K. Strike, K. E. Emam, and N. Madhavji. Software cost estimation with incomplete data. *IEEE Transactions on Software Engineering*, 27(10):890–908, 2001.

[31] N. Suguna and K. G. Thanushkodi. Predicting missing attribute values using k-means clustering. *Journal of Computer Science*, 7(2):216–224, 2011.

[32] B. F. Teasley, L. M. Leventhal, C. R. Mynatt, and D. S. Rohlman. Positive test bias in software engineering professionals: What is right and what's wrong. In *Proceedings of the 5th Workshop on Empirical Studies of Programmers.*, Palo Alto, CA, December 1993.

[33] A. Thobbi and W. Sheng. Imitation learning of arm gestures in presence of missing data for humanoid robots. In *Humanoids'10*, pages 92–97, 2010.

[34] A. Tosun, B. Turhan, and A. Bener. Ensemble of software defect predictors: A case study. In *Proceedings of the 2nd International Symposium on Empirical Software Engineering and Measurement*, Kaiserslautern, Germany, October 2008.

[35] A. Tosun, B. Turhan, and A. Bener. Practical considerations in deploying ai for defect prediction: A case study within the turkish telecommunication industry. In *Proceedings of the 5th International Conference on Predictor Models in Software Engineering*, Vancouver, BC, Canada, May 2009.

[36] W. M. Trochim and J. P. Donnelly. *The Research Methods Knowlege Base*. Atomic Dog/Cengage Learning, 2006.

[37] B. Twala, M. C. , and M. Shepperd. Ensemble of missing data techniques to improve software prediction accuracy. In *Proceedings of the 28th International Conference on Software Engineering*, pages 84–89. ACM, 2006.

[38] P. Wason. On the failure to eliminate hypotheses in a conceptual task. *Quarterly Journal of Experimental Psychology*, 12:129–140, June 1968.

[39] P. Wason. Reasoning about a rule. *Quarterly Journal of Experimental Psychology*, 20:273–281, June 1968.

[40] E. J. Weyuker, T. J. Ostrand, and R. M. Bell. Using developer information as a factor for fault prediction. In *Proceedings of the 1st International Workshop on Predictor Models in Software Engineering*, pages 1–7, Feb. 2007.