Software Qual J (2006) 14: 85–111
DOI 10.1007/s11219-006-7597-z

An empirical study of predicting software faults with case-based reasoning

Taghi M. Khoshgoftaar · Naeem Seliya · Nandini Sundaresh

Abstract The resources allocated for software quality assurance and improvement have not increased with the ever-increasing need for better software quality. A targeted software quality inspection can detect faulty modules and reduce the number of faults occurring during operations. We present a software fault prediction modeling approach with case-based reasoning (CBR), a part of the computational intelligence field focusing on automated reasoning processes. A CBR system functions as a software fault prediction model by quantifying, for a module under development, the expected number of faults based on similar modules that were previously developed. Such a system is composed of a similarity function, the number of nearest neighbor cases used for fault prediction, and a solution algorithm. The selection of a particular similarity function and solution algorithm may affect the performance accuracy of a CBR-based software fault prediction system. This paper presents an empirical study investigating the effects of using three different similarity functions and two different solution algorithms on the prediction accuracy of our CBR system. The influence of varying the number of nearest neighbor cases on the performance accuracy is also explored. Moreover, the benefits of using metric-selection procedures for our CBR system is also evaluated. Case studies of a large legacy telecommunications system are used for our analysis. It is observed that the CBR system using the Mahalanobis distance similarity function and the inverse distance weighted solution algorithm yielded the best fault prediction. In addition, the CBR models have better performance than models based on multiple linear regression.

Keywords Software quality · Case-based reasoning · Software fault prediction · Similarity functions · Solution algorithm · Software metrics

T. M. Khoshgoftaar (✉)
Empirical Software Engineering Laboratory, Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431 USA
e-mail: taghi@cse.fau.edu

1. Introduction

Failure-free operations of software-based systems is often influenced by the quality and reliability of their underlying software. Assuring the desired software quality often demands time-consuming and expensive development processes. Software quality assurance strategies may include rigorous design and code reviews (Porter et al., 1997), automatic test-case generation (Korel, 1996), extensive software testing (Perry, 2000), strategic assignment of key personnel, and re-engineering of the low-quality components of a system. However, such strategies require that a specific amount of resources be allocated primarily for software quality improvement. However, in software engineering practice, the resources allocated for software quality assurance and improvement have not increased with the ever-increasing need for better software quality.

A targeted software quality inspection and improvement can detect faulty software modules and reduce the number of faults occurring during system operations. A software fault is a defect in an executable product that causes system failures during operations. A module is the lowest level of software for which we have data. Software metrics have been shown to be good indicators of software quality (Fenton and Pfleeger, 1997; Gray and MacDonell, 1999; Schneidewind, 2002), which is often represented by a quantitative software quality factor such as the number of faults in a software module. Software metrics which represent the software product and its process can be collected relatively earlier (in the software life cycle) than the software quality factor. Hence, for a currently under-development project the software metrics data may be known, but its software quality data is not known. A software fault prediction model is built using a previously developed project or release for which the software metrics and software quality data is known (Gokhale and Lyu, 1997; Khoshgoftaar et al., 1995; Khoshgoftaar and Seliya, 2002; Troster and Tian, 1995). The model can be applied to predict the number of software faults in the modules of a currently under-development software project.

Software fault prediction models are desired by the software quality team when they are interested in quantifying the quality of a software module. Such a model is of importance from a practical software quality improvement point of view, because based on such a prediction the software quality team can simply prioritize their enhancement efforts starting with the most faulty modules. Consequently, according to the allocated software quality improvement resources, a set of the most faulty modules can be subjected to enhancements. In our recent research effort we introduced a preliminary *Case-Based Reasoning* (CBR) technique for software fault prediction (Ganesan et al., 2000), and compared its performance with that of multiple linear regression (Berenson et al., 1983) using a small-scale case study. This empirical study continues our on-going research on investigating and improving software fault prediction models using CBR. The CBR methodology has also been explored for various problems in the software engineering field, including software cost estimation (Briand et al., 2000; Idri et al., 2002; Kadoda et al., 2000; Shepperd and Schofield, 1997), software reuse (Ramamoorthy et al., 1993), software design (Bartsch-Spoerl, 1995; Bell et al., 1994; Smith and Ganesan, 1995), software quality (Ganesan et al., 2000; Imam et al., 2001), and software help desk (Kriegsman and Barletta, 1993).

A CBR system can be viewed as an automated reasoning approach which obtains (in the specified problem domain) a solution for a current case based on the knowledge of instances of past cases (Leake, 1996). The known knowledge of the past cases is stored in a case library, and represents the experience of the organization within the problem domain. For example, in the context of software fault prediction, the known software metrics and software fault data (of a previously developed project or release) that will be used for analysis is stored in the

case library. In addition, a current or target case represents a software module whose software measurement data is known, but its fault data is unknown. In the case of the CBR models presented in this study for software fault prediction, the working hypothesis is that a software module currently under development will be as faulty as the previously developed (in the case library) software modules with similar software metrics. The similarity or distance between the current and previously developed modules is evaluated by a *similarity function*. Subsequently, a *solution algorithm* is used to predict the number of faults in the target module. There are different types of similarity functions and solution algorithms, as shown in Section 2.

The similarity function and solution algorithm used by the CBR system play an important part in the obtained fault prediction accuracy. Furthermore, the number of nearest neighbor cases (those that are most similar to the target case) used in predicting the response variable (number of faults) of the target module, may also affect the prediction accuracy. Our previous preliminary work with CBR fault prediction models (Ganesan et al., 2000) used only one similarity measure and utilized a very simple solution algorithm.

In this paper, we present an empirical investigation exploring the effects of incorporating different similarity functions (three techniques) and solution algorithms (two methods) on the software fault prediction accuracy of our CBR software quality modeling system (Sundaresh, 2001). In addition, the influence of varying the number of nearest neighbors on the prediction accuracy will be empirically studied. Moreover, in this study we present the *Mahalanobis* distance as a similarity function for CBR-based software quality estimation. Additional details regarding the different types of similarity functions and solution algorithms that we have used are presented in Section 2. To our knowledge, this is the first study to provide a large-scale comprehensive study of CBR-based software fault prediction models. We present four case studies of software metrics and fault data collected over four successive releases of a large legacy telecommunications system.

The first study includes building models with 28 software metrics, which include 24 product and 4 execution metrics. A statistical test using two-way analysis of variance (ANOVA) models is done to verify if the different similarity measures and solutions algorithms yield significantly different results (Berenson et al., 1983). The second study includes building models with the 4 execution metrics and 6 domain metrics extracted by performing principle components analysis (PCA) on the 24 product metrics. This was done to observe if there was any significant improvement in the prediction accuracy of our CBR software fault prediction system.

The third study includes building models with 7 significant metrics obtained by performing the stepwise regression model selection technique (Berenson et al., 1983). This was done to investigate whether the removal of insignificant metrics would result in similar or better prediction accuracy. A Z-test indicates whether the metric-selection process yielded better prediction accuracies. The fourth study consists of running a greedy selection algorithm for addressing the feature subset selection problem in CBR systems. The correlation-based feature subset selection (CFSS) evaluation with greedy stepwise search was used (Hall and Smith, 1998). This technique uses the predictive ability of each attribute in a subset as well as the redundancy between them to evaluate the subset. A subset with high predictive ability and low redundancy is preferred. A forward greedy search is used beginning with no attributes and terminating when the addition of a subset of attributes yields a decrease in the evaluation. This algorithm is implemented in the freeware data mining tool, WEKA (Whitten and Frank, 2000).

The CBR-based software fault prediction models developed for the telecommunications system considered in our study indicated that the Mahalanobis distance similarity function and the inverse distance weighted average solution algorithm will yield the best fault prediction accuracy. It was also observed that models based on feature subset selection obtained by stepwise regression model selection (Berenson et al., 1983) and correlation-based feature

selection with a greedy approach (Hall and Smith, 1998; Whitten and Frank, 2000) did not provide any conclusive improvements in the prediction accuracy of the CBR system. In addition to the case studies presented in this paper, we have performed case studies of other large-scale software systems (Sundaresh, 2001), and similar empirical results as those presented in this paper were observed. A comprehensive comparative study of the CBR-based fault prediction models with those based on other techniques, such as neural networks and regression trees, is out of scope for this study. However, we do compare the CBR-based models with those built using multiple linear regression (Berenson et al., 1983).

The remainder of the paper continues with a description of our CBR system in Section 2. Section 3 discusses the approach adopted in building and evaluating the prediction models. The case study and the empirical analysis are presented in Sections 4 and 5, respectively. We conclude in Section 6 with a summary and suggestions for future research.

2. Case-based reasoning

A CBR system has several practical advantages over other software quality prediction techniques:

- *Case Alert*: CBR systems can be designed to alert users when a new case is outside the bounds of current experience. Instead of guessing a solution, an answer of "I don't know" is usually better. In contrast to CBR, other modeling techniques may provide some kind of solution even in extreme cases.
- *Model Adaptivity*: As new information becomes available, cases can be added to or deleted from the case library without the need for model re-calibration. This is an attractive property of CBR systems, because other techniques may require frequent re-estimation of model parameters to incorporate the changes in available data.
- *Model Scalability*: CBR systems are scalable to very large case libraries, and fast retrieval of cases continues to be of practical advantage.
- *Model Interpretation*: Once the most relevant case(s) is(are) selected the detailed description, including quantitative attributes of the case(s) can assist in the interpretation of the automated estimation.
- *User Acceptance*: CBR systems are not "black boxes." One can be easily convinced that a given solution was derived in a reasonable way, and hence, the CBR system lends itself to user acceptance.

Generally speaking, a CBR prediction system consists of a case library, a solution algorithm, a similarity function, and the associated retrieval and decision rules (Kolodner, 1993; Leake, 1996). The case library consists of well-known project data from a previously developed system release or similar project, and contains all relevant information pertaining to each case (software module). In the context of software quality estimation, such cases are composed of a set of software metrics or independent variables ($\mathbf{x}_i$), and a response or dependent variable (y_i). In our study of software fault prediction, y_i represents the number of faults in module i. The software metrics collected during the software development process are often measured in varying ways and could contain a variety of ranges and scales. Therefore, before using the data for modeling purposes, it should be *standardized* (Berenson et al., 1983).

A solution algorithm uses a similarity function to estimate the relationship or distance between the target case (whose dependent variable is to be estimated) and each case in the case library. Subsequently, the algorithm retrieves relevant cases and determines a solution to the new problem, i.e., it predicts the y_i value of the target case. A *similarity function*

determines, for a given case, the most similar cases from the case library (*fit* or *training* data). The function computes the distance d_{ij} between the current (or target) case $\mathbf{x}_i$ and every case $\mathbf{c}_j$ in the case library. In our study we considered three types of similarity functions, namely, *City Block distance*, *Euclidean distance*, and *Mahalanobis distance*.

The cases with the smallest possible distances are of primary interest and the set of similar cases forms the set of *nearest neighbors*, N. The CBR model parameter n_N, represents the number of the best (most similar to current or target case) cases selected from N for further analysis. The parameter n_N can be varied during model calibration to obtain different models. Once n_N is selected, a solution algorithm is used to predict the dependent variable. In the case of software fault prediction modeling, we used the *unweighted average* and *inverse distance weighted average* solution algorithms. We now discuss the details of the three similarity functions and two solution algorithms considered in our study.

2.1. Similarity functions

2.1.1. City block distance

This similarity function is also known as *Absolute distance* or *Manhattan distance*. It is computed by taking the weighted sum of the absolute value of the difference in independent variables between the current case and a past case (from the case library). The weight associated with each independent variable is provided by the user or the analyst. This distance function is primarily used for numeric attributes, and is given by:

$$d_{ij} = \sum_{k=1}^{m} w_k |x_{ik} - c_{jk}| \tag{1}$$

where m is the number of independent variables, and w_k is the weight of the k^{th} independent variable. In our study, $w_k = 1$ for the City Block distance and the Euclidean distance similarity measures.

2.1.2. Euclidean distance

This similarity function views the independent variables as dimensions within an m-dimensional space, with m being the number of independent variables. A current case is represented as a point within this space. The distance is calculated by taking the weighted distance between the current case and a past case within this space. This distance function is also commonly used when the data set contains quantitative attributes, and is given by:

$$d_{ij} = \sqrt{\sum_{k=1}^{m} (w_k (x_{ik} - c_{jk}))^2} \tag{2}$$

2.1.3. Mahalanobis distance

This distance measure is an alternative to the Euclidean distance. It is used when the independent variables are *highly correlated*. The Mahalanobis distance is a very attractive similarity function to implement because it can explicitly account for the correlation among

the software attributes (Fenton and Pfleeger, 1997), and the independent variables do not need to be standardized.[1] It is given by (Dillon and Goldstein, 1984):

$$d_{ij} = (\mathbf{x}_i - \mathbf{c}_j)' S^{-1} (\mathbf{x}_i - \mathbf{c}_j) \tag{3}$$

where ($'$) implies transpose, S is the variance-covariance matrix of the independent variables over the entire case library, and S^{-1} is its inverse.

2.2. Solution algorithms

2.2.1. Unweighted average

This algorithm estimates the dependent variable, $\widehat{y_i}$, by calculating the average of the number of faults of the most similar modules, i.e., n_N, from the case library. The predicted value is therefore given by:

$$\widehat{y_i} = \frac{1}{n_N} \sum_{j \in N} y_j \tag{4}$$

2.2.2. Inverse distance weighted average

The dependent variable is estimated using the distance measures for the n_N closest cases as weights in a weighted average. Since smaller distances indicate a closer match, each case is weighted by a normalized inverse distance. The case most similar to the target module has the largest weight, thus playing a major role in the obtained fault prediction.

$$\delta_{ij} = \frac{1/d_{ij}}{\sum_{j \in N} 1/d_{ij}} \tag{5}$$

$$\widehat{y_i} = \sum_{j \in N} \delta_{ij} y_j \tag{6}$$

2.3. Our previous CBR-related work

We have conducted prior studies based on our CBR approach. In (Khoshgoftaar and Seliya, 2003), we investigate a CBR-based approach to software quality classification. In contrast, this paper presents a CBR-based approach for software fault (quantitative) prediction.

An outlier detection approach for improving a CBR-based software quality classification models was presented in (Khoshgoftaar et al., 2003). The aim of that study was to generate rules from the dataset to identify and eliminate outliers in the context of a CBR classifier.

In a recent study (Khoshgoftaar et al., 2003), a detailed investigation was made on feature subset selection based on the Kolomogorov-Smirnov two-sample statistical test for a software measurement dataset. The selected features were then used to build a CBR-based software quality classification model.

[1] Standardization is defined by $\frac{x_k - \overline{x}_k}{s_k}$, where $\overline{x}_k$ is the mean and s_k is the standard deviation of the k^{th} metric.

The solution algorithms in our CBR-based models used for classification and quantitative prediction are different. In (Khoshgoftaar et al., 1997), a simple matching strategy determined the class of the target instance. In this paper, we use the unweighted and inverse distance weighted averages to predict the number of faults in the target module.

3. Methodology

The complete CBR modeling process for building software fault prediction models has been implemented and automated in an empirical software quality modeling tool, the Software Measurement Analysis and Reliability Toolkit (SMART) (Khoshgoftaar et al., 2000). The tool, developed by our research team at the Empirical Software Engineering Laboratory, Florida Atlantic University, is a Windows©-based application implemented in Visual C++© and has a user-friendly graphical interface. The modeling capability of the current version of SMART includes building and evaluating CBR models for software fault prediction (Sundaresh, 2001) and software quality classification (Khoshgoftaar and Seliya, 2003).

3.1. Building and evaluating models

1. *Case Library and Target Data*: The CBR models are to be calibrated for three case studies of the legacy telecommunications system. The three case studies are described in detail in Section 4, and include a study with the original 28 software metrics, a study with the domain metrics extracted by principle components analysis, and a study with a reduced data set obtained by the stepwise regression model selection technique. The software metrics of the original data set are standardized to a mean of zero and a variance of one. This is done so that all the metrics are converted to a uniform system of co-ordinates with the same unit of measure. However, standardization is not required when the Mahalanobis distance function is used as a similarity measure. In each of the studies, Release 1 is used as the fit data set, i.e., the case library. The other three releases, i.e., Release 2, Release 3, and Release 4 are used as test data sets.
2. *Similarity Function and Solution Algorithm*: The desired similarity function, i.e., City Block, Euclidean, or Mahalanobis, is selected for modeling. Furthermore, the desired solution algorithm, i.e., unweighted average or inverse distance weighted average, is selected.
3. *Parameter* n_N: The number of nearest neighbor cases, n_N, to be used by the solution algorithm for estimating the number of faults is varied as a model parameter. Hence, for each value of n_N a unique model is built, as discussed in the next step. The values that we considered for n_N ranged from 1 to 100. Other values for n_N are also considered, but did not reveal results of any significance.
4. *Building Models*: The models are calibrated using the *leave-one-out* (LOO) cross-validation technique, which is described as follows. Assume a *fit* data set, i.e., case library, with q cases. During each iteration one case is removed and the model is trained using the $q - 1$ cases. The trained model is then used to predict the number of faults for the removed case, which acts as the test or target case. The *absolute* and *relative* error values (see Section 3.2) are computed for the prediction of the test case. Hence, q iterations are performed such that the fault prediction for a case, v, is based on a model that is trained using all the cases in the fit data set, except v. At the end of q iterations, a number of faults estimation for all the cases in the fit data set is obtained. The *average absolute error* (AAE) and *average relative error* (ARE) for the fit data are computed based on the LOO prediction estimates.

5. *Model Selection:* The trained CBR model whose n_N resulted in the minimum AAE (using the LOO technique) is selected as the preferred CBR fault prediction model. Among the CBR models with the lowest AAE values, the one with the smallest n_N is selected as the preferred model. The ARE performance measure is not used for model selection purposes, because in the context of a similar prediction problem, recent studies have pointed out the ambiguities of using ARE as a performance (predictive capability) metric (Shepperd and Kadoda, 2001). However, we present the ARE values of the different models that were built.
6. *Evaluating Models*: The prediction accuracy of the CBR models is expressed using the *test* data sets that are independent of the fit data set. The use of an independent data set for evaluating prediction accuracy simulates the application of a model on a currently under-development system release. The AAE and ARE performance measures are computed for the test data sets, and the respective models are studied.

3.2. Performance evaluation

The prediction accuracy of the models are determined by computing the two commonly used performance measures, i.e., AAE and ARE.

$$\text{AAE} = \frac{1}{n}\sum_{i=1}^{n} |y_i - \widehat{y}_i| \tag{7}$$

$$\text{ARE} = \frac{1}{n}\sum_{i=1}^{n} \left| \frac{y_i - \widehat{y}_i}{y_i + 1} \right| \tag{8}$$

where n is the number of modules in the given (fit or test) data set. The denominator in ARE has a '1' added to avoid division by zero. We have presented and utilized both measures, because the comparative study of AAE and ARE as performance metrics is out of scope for this paper.

3.3. Analysis of variance models

We investigate whether the fault prediction models based on a particular similarity function or solution algorithm yielded significantly better results than models based on the other similarity functions and solution algorithms. This is done by building two-way analysis of variance (ANOVA) models (Berenson et al., 1983). The three similarity functions and two solution algorithms provide six unique combinations.

The ANOVA models will assess if the differences in the predictions, i.e., AAE and ARE, obtained by the six combinations are of statistical significance. A two-way ANOVA model with replication is used: factor A is the similarity function, factor B is the solution algorithm, and response variable is the error value, absolute error or relative error. The null hypothesis indicated there is no significant difference between the performance of the respective factors. The ratios of mean squares of factors to mean squared error are distributed according to F-distributions (Berenson et al., 1983).

4. System description and case studies

The software metrics and fault data for this case study (denoted as LLTS) were collected over four historical releases from a very large legacy telecommunications system. The software system is written in a high-level language using the procedural paradigm, and maintained

by professional programmers in a large organization. The system had significantly over ten million lines of code and included numerous finite-state machines and interfaces to various kinds of equipment. We labeled the four releases 1 through 4. These releases were the last four releases of the legacy system. The modules of Release 1 were used as the *fit* data set for training the software quality estimation models. The case library of our CBR system represents the fit data set for the given case study. The respective modules of Releases 2, 3, and 4 are used as the three *test* data sets for evaluating the predictive performances of the models.

A software module was considered as a set of functionally related source-code files according to the system's architecture. Fault data collected at the module-level by the problem reporting system consisted of faults discovered during post unit testing phases, and was recorded before and after the product was released to customers. A fault was recorded and attributed to a module if its source code was modified to fix a given problem. The software faults prior to deployment included faults discovered by designers, faults discovered during beta testing, and any other problems that initiated a change in the source code. The software faults discovered by customers in the field (external) were recorded only when their discovery resulted in a change in the source code, and included all types of software faults. The number of modules associated with faults were very few compared to modules with no faults, as shown in Figure 1. This is reflective of the nature of the system being modeled, i.e., a high-assurance system. Two clusters of modules were identified: unchanged and updated.

The updated modules consisted of those that were either new or had at least one update to their source code since the prior release. Configuration management data analysis identified software modules that were unchanged from the prior release. Among the unchanged modules, almost all of them had no faults. Consequently, in our studies we considered only modules that were new, or had at least one source code update since the prior release. The distribution of the post unit testing faults was such that most modules of the legacy system had no faults associated with them. The system had several million lines of code in a few thousand modules per release. Each release had approximately 3500 to 4000 updated or new software modules. The number of modules considered in Releases 1, 2, 3, and 4 were 3649, 3981, 3541, and 3978 respectively.

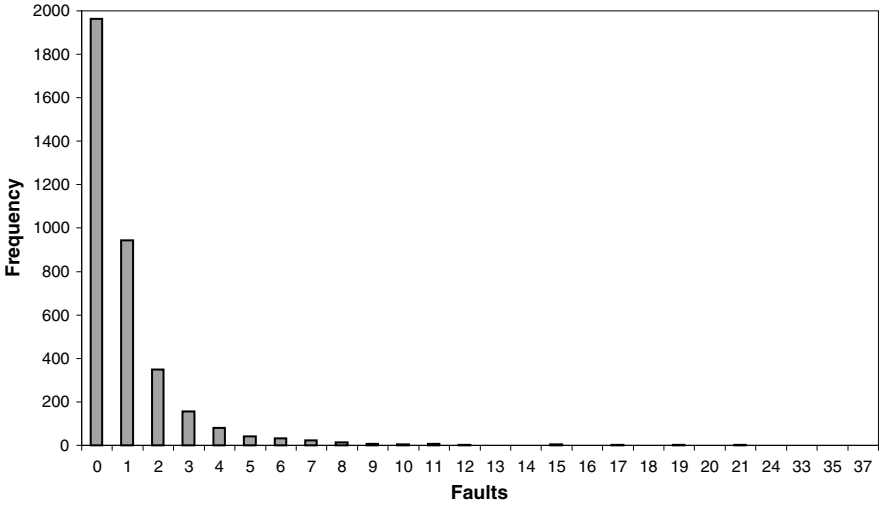

Fig. 1 Distribution of faults for Release 1

The set of available software metrics is usually determined by pragmatic considerations. A data mining approach is preferred in exploiting software metrics data (Fayyad, 1996), by which a broad set of metrics are analyzed rather than limiting data collection according to a predetermined set of research questions. Data collection for this case study involved extracting source code from the configuration management system. The available data collection tools determined the number and selection of the software metrics. Software measurements were recorded using the EMERALD (Enhanced Measurement for Early Risk Assessment of Latent Defects) software metrics analysis tool, which includes software-measurement facilities and software quality models (Hudepohl et al., 1996).

Preliminary data analysis selected metrics (aggregated at the module level) that were appropriate for our modeling purposes. The software metrics considered in our studies included 24 product metrics, 14 process metrics, and 4 execution metrics. The 14 process metrics were not used for modeling purposes, because we (and the software project management team) are concerned with a timely software fault prediction after the implementation (coding) phase and prior to system tests and operations. It should be noted that the software metrics used for the legacy system may not be universally appropriate for all software systems. Another project might collect (depending on availability) and use a different set of software metrics.

The software product metrics in Table 1 are based on call graph, control flow graph, and statement metrics. A module's call graph depicts the calling relationships among procedures. A module's control flow graph consists of nodes and arcs depicting the flow of control of the program. Statement metrics are measurements of the program statements without expressing the meaning or logic of the statements. The execution metrics listed in Table 2 are associated with the likelihood of executing a module, i.e., operational use. The proportion of installations that had a module, *USAGE*, was approximated by deployment data on a prior release. Execution times were measured in a laboratory setting with different simulated workloads. The three case studies that were performed for the legacy telecommunications system are described below.

1. The first case study, denoted RAW-28, consisted of building models with all 28 software metrics (independent variables) described above. More specifically, the RAW-28 case study utilized the 24 product and 4 execution metrics to predict the number of faults in a module during post unit testing.
2. The second case study, denoted PCA-10, consisted of building models with the 4 software execution metrics and the 6 domain metrics that were extracted by performing principle components analysis on the 24 software product metrics. The aim of the PCA-10 case study is to investigate whether the removal of correlation among the software product attributes yielded in any improvements in the fault prediction accuracy of our CBR system.
3. The third case study, denoted RAW-7, consisted of building models with the 7 significant metrics (out of the 28) that were obtained by performing a stepwise regression analysis on the original data set, i.e., RAW-28. The aim of the RAW-7 case study is to investigate, in the context of our CBR system, whether using a smaller set of software metrics for the legacy system would yield similar or better fault prediction results.

5. Results and analysis

5.1. Determining optimum n_N

The number of nearest neighbor cases (n_N) used by the CBR system for obtaining the required solution is a parameter that can affect the prediction accuracy. Determining the optimum value

Table 1 Software product metrics

Symbol	Description
	Call Graph Metrics
CALUNQ	Number of distinct procedure calls to others.
CAL2	Number of second and following calls to others.
	$CAL2 = CAL - CALUNQ$ where *CAL* is the total number of calls.
	Control Flow Graph Metrics
CNDNOT	Number of arcs that are not conditional arcs.
IFTH	Number of non-loop conditional arcs, ie, if-then constructs.
LOP	Number of loop constructs.
CNDSPNSM	Total span of branches of conditional arcs. The unit of measure is arcs.
CNDSPNMX	Maximum span of branches of conditional arcs.
CTRNSTMX	Maximum control structure nesting.
KNT	Number of knots. A "knot" in a control flow graph is where arcs cross due to a violation of structured programming principles.
NDSINT	Number of internal nodes (i.e., not an entry, exit, or pending node).
NDSENT	Number of entry nodes.
NDSEXT	Number of exit nodes.
NDSPND	Number of pending nodes, i.e., dead code segments.
LGPATH	Base 2 logarithm of the number of independent paths.
	Statement Metrics
FILINCUQ	Number of distinct include files.
LOC	Number of lines of code.
STMCTL	Number of control statements.
STMDEC	Number of declarative statements.
STMEXE	Number of executable statements.
VARGLBUS	Number of global variables used.
VARSPNSM	Total span of variables.
VARSPNMX	Maximum span of variables.
VARUSDUQ	Number of distinct variables used.
VARUSD2	Number of second and following uses of variables.
	$VARUSD2 = VARUSD - VARUSDUQ$ where *VARUSD* is the total number of variable uses.

Table 2 Software execution metrics

Symbol	Description
USAGE	Deployment percentage of the module.
RESCPU	Execution time (microseconds) of an average transaction on a system serving consumers.
BUSCPU	Execution time (microseconds) of an average transaction on a system serving businesses.
TANCPU	Execution time (microseconds) of an average transaction on a tandem system.

Table 3 RAW-28, optimum n_N with euclidean distance

	Unweighted		Weighted	
n_N	AAE	ARE	AAE	ARE
1	1.152	0.560	1.152	0.560
3	1.081	0.548	1.022	0.509
5	1.042	0.538	0.983	0.496
7	1.024	0.534	0.966	0.492
9	1.016	0.531	0.953	0.486
11	1.011	0.532	0.947	0.485
13	1.009	0.534	0.945	0.486
15	1.002	0.532	0.938	0.484
17	1.000	0.532	0.935	0.483
19	**0.995**	**0.530**	0.930	0.481
21	0.995	0.530	0.929	0.480
23	0.998	0.531	0.931	0.481
25	0.996	0.530	0.929	0.480
27	0.996	0.531	0.928	0.480
29	0.996	0.531	0.928	0.479
31	0.996	0.531	0.928	0.479
33	1.057	0.607	**0.927**	**0.479**
35	1.057	0.607	0.928	0.479
37	1.057	0.607	0.928	0.479
39	1.057	0.607	0.928	0.479
41	1.057	0.607	0.928	0.479
43	1.057	0.607	0.928	0.479
45	1.057	0.607	0.928	0.479
47	1.057	0.607	0.929	0.474

of n_N is dependent on the case study data set, and hence, requires an empirical approach. For a given similarity function and solution algorithm, the trained model whose n_N yields the minimum AAE (using LOO) is selected as the preferred CBR model.

The following two sub-sections present the variation of AAE and ARE with respect to n_N for the RAW-28 and PCA-10 case studies. It should be noted that not all (odd and even) results from values of n_N we have considered are included in the paper. For presentation and paper-size purposes, the tables in this section (Tables 3, 4, 5, 6, 7, and 8) report only some of the odd values that were considered for n_N. However, we have presented a good spectrum (from either end) of values covering the selected n_N.

5.1.1. RAW-28 case study

Table 3 presents the AAE and ARE values when the unweighted average and inverse distance weighted average solution algorithms are used in combination with the Euclidean distance similarity measure. Along the same lines Tables 4 and 5, respectively, present the same for the

Table 4 RAW-28, optimum n_N with city block distance

	Unweighted		Weighted	
n_N	AAE	ARE	AAE	ARE
1	1.147	0.554	1.147	0.554
3	1.060	0.538	1.008	0.503
5	1.031	0.530	0.971	0.488
7	1.015	0.527	0.951	0.483
9	1.011	0.530	0.945	0.483
11	1.012	0.532	0.944	0.484
13	1.004	0.529	0.938	0.481
15	1.000	0.527	0.934	0.480
17	0.994	0.526	0.928	0.478
19	0.999	0.528	0.930	0.479
21	0.994	0.525	0.926	0.476
23	**0.991**	**0.523**	0.925	0.475
25	1.055	0.600	0.924	0.474
27	1.055	0.600	0.924	0.474
29	1.054	0.599	0.924	0.474
31	1.053	0.599	0.924	0.474
33	1.051	0.598	0.923	0.473
35	1.049	0.597	**0.922**	**0.473**
37	1.051	0.598	0.923	0.473
39	1.049	0.598	0.924	0.474
41	1.057	0.607	0.924	0.474
43	1.057	0.607	0.925	0.474
45	1.057	0.607	0.925	0.474
47	1.057	0.607	0.926	0.475

Table 5 RAW-28, optimum n_N with Mahalonobis distance

	Unweighted		Weighted	
n_N	AAE	ARE	AAE	ARE
1	1.010	0.455	1.010	0.455
3	0.984	0.452	0.936	0.42
5	0.959	0.444	0.905	0.406
7	0.962	0.446	0.902	0.404
9	0.960	0.445	0.900	0.403
11	**0.958**	**0.446**	0.897	0.402
13	0.961	0.449	0.896	0.402
15	0.963	0.449	0.895	0.401
17	0.965	0.450	0.897	0.402
19	0.964	0.448	**0.896**	**0.400**
21	0.965	0.448	0.896	0.399
23	0.966	0.448	0.899	0.400
25	0.966	0.446	0.899	0.399
27	0.967	0.446	0.900	0.398
29	0.967	0.445	0.900	0.398
31	0.968	0.446	0.900	0.398
33	0.968	0.445	0.901	0.398
35	0.968	0.445	0.901	0.397

Table 6 PCA-10, optimum n_N with euclidean distance

	Unweighted		Weighted	
n_N	AAE	ARE	AAE	ARE
61	0.981	0.498	0.917	0.449
63	0.980	0.498	0.917	0.449
65	0.979	0.497	0.916	0.448
67	0.980	0.497	0.916	0.448
69	0.979	0.496	0.916	0.448
71	0.978	0.495	0.915	0.447
73	**0.977**	**0.495**	0.914	0.446
75	0.978	0.494	0.914	0.446
77	0.987	0.494	0.915	0.446
79	0.978	0.493	0.914	0.445
81	0.979	0.493	0.915	0.445
83	0.979	0.493	0.914	0.444
85	0.979	0.493	0.915	0.444
87	0.979	0.492	0.914	0.444
89	0.979	0.492	0.914	0.443
91	0.979	0.492	0.914	0.443
93	0.979	0.491	**0.913**	**0.443**
95	0.979	0.491	0.914	0.443
100	0.978	0.490	0.914	0.442
105	0.978	0.489	0.914	0.441
110	0.978	0.489	0.914	0.441
115	0.978	0.488	0.914	0.440
120	0.979	0.488	0.915	0.440
125	0.979	0.487	0.915	0.440
130	0.980	0.487	0.915	0.439

City Block and Mahalanobis distance similarity measures. The AAE and ARE values shown in the tables are those obtained by the LOO technique as applied to the fit data set, i.e., Release 1. We observe that as the value of n_N is increased, a trend is observed in the values of AAE, i.e., it decreases until a particular n_N is reached, after which it starts to increase with n_N. Consequently, the n_N corresponding to the minimum AAE is chosen as the optimum n_N for our case studies. The respective models selected are highlighted in **bold**. For example, in the case of the models based on the inverse weighted distance average solution algorithm and the Mahalanobis distance similarity measure (Table 5), the preferred model is the one with $n_N = 19$.

5.1.2. PCA-10 case study

The performance metrics observed for this case study are presented in Tables 6, 7, and 8. In the context of the AAE values, a trend similar to the one observed for the RAW-28 case

Table 7 PCA-10, optimum n_N with city block distance

	Unweighted		Weighted	
n_N	AAE	ARE	AAE	ARE
47	0.972	0.485	0.910	0.439
49	0.972	0.485	0.909	0.438
51	0.971	0.483	0.908	0.437
53	0.970	0.483	0.908	0.437
55	0.970	0.482	0.908	0.436
57	0.970	0.481	0.907	0.434
59	0.970	0.481	0.907	0.434
61	**0.968**	**0.481**	0.906	0.433
63	0.968	0.478	0.906	0.432
65	0.968	0.478	0.906	0.432
67	0.968	0.478	**0.905**	**0.432**
69	0.968	0.478	0.906	0.432
71	0.969	0.478	0.906	0.432
73	0.969	0.478	0.906	0.432
75	0.969	0.478	0.906	0.431
77	0.969	0.477	0.906	0.431
79	0.970	0.477	0.906	0.431
81	0.970	0.477	0.907	0.431
83	0.971	0.477	0.907	0.431
85	0.971	0.477	0.907	0.431
87	0.972	0.477	0.908	0.431
89	0.972	0.477	0.908	0.430
91	0.972	0.476	0.908	0.430
93	0.972	0.476	0.908	0.430
95	0.972	0.475	0.908	0.429

study is noted in the PCA-10 case study. However, the minimum AAE value for this case study corresponds to a much larger n_N as compared to that for the RAW-28 case study. When comparing (the corresponding LOO results) the RAW-28 and PCA-10 models, it is observed that lower AAE and ARE values are achieved by the PCA-10 model when either the Euclidean or City Block distances are used as similarity measures with the combination of either the unweighted average or inverse distance weighted average solution algorithm. However, when the Mahalanobis distance is used in combination with either the unweighted average or inverse distance weighted average solution algorithm, the RAW-28 model depicts better AAE and ARE values (for the fit data set) than the PCA-10 model.

5.2. Model evaluation for RAW-28 case study

Table 9 summarizes the AAE and ARE values (and their standard deviations) of the respective CBR models across the different system releases. Among the models calibrated by using

Table 8 PCA-10, optimum n_N with Mahalonobis distance

	Unweighted		Weighted	
n_N	AAE	ARE	AAE	ARE
47	0.979	0.497	0.920	0.451
49	0.979	0.497	0.919	0.451
51	0.980	0.496	0.919	0.450
53	0.980	0.497	0.919	0.450
55	0.979	0.495	0.917	0.449
57	0.979	0.496	0.917	0.448
59	0.979	0.495	0.917	0.448
61	0.979	0.495	0.916	0.447
63	0.978	0.494	0.915	0.446
65	0.978	0.493	0.915	0.446
67	0.978	0.493	0.914	0.446
69	0.978	0.493	0.914	0.445
71	0.977	0.493	0.914	0.445
73	0.977	0.492	0.913	0.444
75	**0.976**	**0.491**	0.913	0.444
77	0.976	0.491	0.913	0.444
79	0.976	0.491	0.913	0.444
81	0.976	0.490	**0.912**	**0.443**
83	0.976	0.489	0.912	0.442
85	0.977	0.490	0.913	0.442
87	0.978	0.490	0.913	0.442
89	0.978	0.490	0.913	0.442
91	0.978	0.490	0.913	0.442
93	0.978	0.489	0.913	0.441
95	0.978	0.489	0.913	0.440

different similarity functions and solution algorithms, the ones with the lowest AAE values are shown in the table. The error values (AAE and ARE) for Release 1 are based on the LOO cross-validation technique. The selected model is then applied to the test data sets, i.e., Releases 2, 3, and 4, and the respective error statistics are computed. It is observed that for the RAW-28 case study, the Mahalanobis distance measure has the lowest AAE and ARE values (with both solution algorithms). This suggests that it yielded a better fault prediction accuracy as compared to the Euclidean and City Block distance similarity measures. Furthermore, for almost all (11 out of 12) the cases shown in Table 9, the inverse distance weighted average solution algorithm yielded a lower AAE value.

5.3. Model evaluation for PCA-10 case study

Similar to the Table 9 provided for the RAW-28 case study, Table 10 presents the performance metrics for the PCA-10 case study. The table indicates that for the majority of the cases, the

Table 9 Selected models for RAW-28 case study

Data set	Similarity function	Unweighted average				Weighted average			
		AAE	SDAE*	ARE	SDRE*	AAE	SDAE	ARE	SDRE
Release 1*	Euclidean	0.995	1.618	0.530	0.537	0.927	1.635	0.479	0.513
Release 1	CityBlock	0.991	1.619	0.523	0.520	0.922	1.642	0.473	0.514
Release 1	Mahalonobis	0.958	1.745	0.446	0.394	0.896	1.762	0.399	0.345
Release 2	Euclidean	0.904	1.019	0.595	0.589	0.884	1.100	0.585	0.562
Release 2	CityBlock	0.898	1.020	0.590	0.580	0.885	0.994	0.584	0.556
Release 2	Mahalonobis	0.824	1.048	0.490	0.425	0.879	0.980	0.528	0.551
Release 3	Euclidean	0.955	1.142	0.598	0.633	0.926	1.100	0.581	0.572
Release 3	CityBlock	0.950	1.135	0.594	0.616	0.927	1.090	0.581	0.568
Release 3	Mahalonobis	0.869	1.153	0.495	0.418	0.861	1.119	0.499	0.456
Release 4	Euclidean	0.944	1.030	0.588	0.607	0.925	0.998	0.576	0.571
Release 4	CityBlock	0.943	1.037	0.588	0.620	0.925	1.091	0.581	0.568
Release 4	Mahalonobis	0.846	1.002	0.502	0.473	0.831	0.981	0.492	0.426

*SDAE denotes standard deviation of absolute error
*SDRE denotes standard deviation of relative error
*Release 1 was fit data set

Table 10 Selected models for PCA-10 case study

Data set	Similarity function	Unweighted average				Weighted average			
		AAE	SDAE*	ARE	SDRE*	AAE	SDAE	ARE	SDRE
Release 1	Euclidean	0.977	1.695	0.495	0.401	0.913	1.714	0.443	0.393
Release 1	CityBlock	0.968	1.702	0.479	0.370	0.905	1.713	0.432	0.377
Release 1	Mahalonobis	0.976	1.698	0.491	0.391	0.912	1.707	0.443	0.411
Release 2	Euclidean	0.853	1.671	0.488	0.397	0.844	1.000	0.532	0.423
Release 2	CityBlock	0.842	1.012	0.523	0.408	0.835	0.986	0.523	0.406
Release 2	Mahalonobis	0.852	1.023	0.533	0.426	0.841	0.964	0.539	0.453
Release 3	Euclidean	0.885	1.147	0.532	0.448	0.878	1.127	0.528	0.434
Release 3	CityBlock	0.874	1.138	0.519	0.420	0.871	1.115	0.519	0.419
Release 3	Mahalonobis	0.884	1.147	0.530	0.441	0.879	1.105	0.533	0.458
Release 4	Euclidean	0.803	0.975	0.479	0.301	0.801	0.973	0.478	0.294
Release 4	CityBlock	0.812	0.986	0.476	0.285	0.810	0.986	0.477	0.287
Release 4	Mahalonobis	0.805	0.975	0.482	0.298	0.807	0.972	0.488	0.311

City Block similarity measure performs (except for Release 4) better, i.e., lower AAE, than the Euclidean and Mahalanobis distance measures. However, on performing a Z-test (not shown) at a significance level of $\alpha = 5\%$ (Berenson et al., 1983), it was observed that the difference was not of statistical significance.

When observing the performances of the solution algorithms for the PCA-10 case study, the inverse distance weighted average solution algorithm generally provides a lower or similar

AAE value than the unweighted average algorithm. We recall that the Mahalanobis distance is used as an alternative to the Euclidean distance when the software attributes are highly correlated. Since PCA removes the correlation among the software metrics, the two measures would provide similar results. This is confirmed by the AAE and ARE values presented in Table 10.

In the context of performance comparisons of the RAW-28 and PCA-10 models, we observe that for all four releases the PCA-10 models yielded better AAE and ARE values when either the Euclidean or City Block distances are used as similarity measures in conjunction with either solution algorithms. However, this is not consistently observed for the models when the Mahalanobis distance is used as the similarity measure. When the Mahalanobis distance is used in combination with the unweighted average solution algorithm, the RAW-28 model depicts better (except for Release 4) AAE and ARE values than the corresponding PCA-10 model.

In contrast, when the Mahalanobis distance is used in combination with the inverse distance weighted average solution algorithm, the RAW-28 model depicts better AAE and ARE values than the corresponding PCA-10 model only for Release 1 and Release 3. Since for a majority of the cases, i.e., different models and different releases, the Mahalanobis distance provides better AAE and ARE values, the use of principle components analysis was not beneficial for the legacy telecommunications system studied. In addition, PCA is not necessary because the Mahalanobis distance measure accounts for the (often observed) high correlation among software metrics.

5.4. Model evaluation for RAW-7 case study

Given a large set of software metrics for software quality prediction purposes, some of them may not contribute significantly to the obtained prediction. In addition, the inclusion of those insignificant metrics during modeling may add noise to the obtained prediction results (Ganesan et al., 2000). Hence, for robust software quality prediction models, only the significant software metrics should be considered for modeling purposes. The proposed CBR prediction system does not support an in-built attribute selection procedure. In fact, most CBR systems do not support an in-built attribute selection procedure, and the feature subset selection is an active research issue for CBR systems (Aha and Bankert, 1994).

We utilized the commonly used stepwise regression technique (Berenson et al., 1983) to determine the significant software metrics among the 28 software metrics described earlier. The stepwise regression technique was performed for a significance level of $\alpha = 0.05$. It was found that the following 7 software metrics were considered significant at a 5% level. The include (see Table 1): *FILINCUQ*, *CNDNOT*, *NDSENT*, *NDSEXT*, *NDSPND*, *NDSINT* and *STMDEC*.

The empirical results obtained from this case study are presented in Table 11, which also summarizes the results of the RAW-28 case study. We note that though all six combinations of similarity functions and solution algorithms were considered during this case study, the table only presents the results of the model selected using the Mahalanobis distance similarity measure when combined with the two solution algorithms. This is because the models based on the Mahalanobis distance yielded the best results in terms of the AAE and ARE values.

A quick inspection of the AAE values shown in Table 11 indicates that neither model clearly outperforms the other for all data sets. To determine whether a significant difference existed between the predictive capabilities of the two models, i.e., RAW-28 and RAW-7, we performed a Z-test. The obtained *p-values*, presented in Table 12, indicate the statistical difference between the two models for the combination of the Mahalanobis similarity measure and the inverse distance weighted average solution algorithm. When comparing the

Table 11 RAW-7 and RAW-28 models with Mahalonobis distance

Case study		Unweighted average				Weighted average			
Model	Data set	AAE	SDAE	ARE	SDRE	AAE	SDAE	ARE	SDRE
RAW-7	Release 1	0.984	1.699	0.493	0.343	0.923	1.707	0.451	0.395
CFSS-7	Release 1	0.987	1.684	0.498	0.420	0.923	1.700	0.439	0.390
RAW-28	Release 1	0.958	1.745	0.446	0.394	0.896	1.762	0.399	0.345
RAW-7	Release 2	0.842	1.020	0.522	0.371	0.846	1.013	0.536	0.440
CFSS-7	Release 2	0.856	1.020	0.531	0.455	0.904	1.209	0.550	0.618
RAW-28	Release 2	0.824	1.048	0.490	0.425	0.879	0.980	0.528	0.551
RAW-7	Release 3	0.861	1.127	0.514	0.391	0.859	1.080	0.523	0.465
CFSS-7	Release 3	0.909	1.129	0.548	0.483	0.927	1.204	0.541	0.586
RAW-28	Release 3	0.869	1.153	0.495	0.418	0.861	1.119	0.499	0.456
RAW-7	Release 4	0.825	0.945	0.503	0.387	0.832	0.944	0.511	0.424
CFSS-7	Release 4	0.878	0.982	0.549	0.498	0.891	1.088	0.526	0.523
RAW-28	Release 4	0.846	1.002	0.502	0.473	0.831	0.981	0.492	0.426

Table 12 *Z*-test results: *p-values*

RAW-7 vs. RAW-28		
Data set	AAE	ARE
Release 1	0.2546	0.00003
Release 2	0.0694	0.23580
Release 3	0.4721	0.01430
Release 4	0.4761	0.47670

prediction accuracy of the two models, we do not consider Release 1 because it was used as the training data set.

We compare the two models at a 10% significance level, and use an adjusted $alpha$ value for the comparison based on the Bonferroni correction. The Bonferroni correction is a multiple-comparison correction used when several tests are performed simultaneously and implies that the *alpha* value needs to be lowered to account for the number of comparisons being performed (Berenson et al., 1983). A conservative approach to do so is to use an *alpha* value divided by the number of comparisons. Since we compare the two models only for their performances on the three test datasets, we evaluate them with respect to one-third of the 10% significance level, i.e., $\alpha = 0.0333$.

It is observed that for the AAE performance measure, there is no statistical difference between the RAW-28 and RAW-7 models for all three test datasets. More specifically, the respective *p-values* are greater than 0.0333. A similar observation was made for the ARE performance measure, except in the case of Release 3 for which $p = 0.0143$. In general, the results of Table 12 do not indicate any conclusive evidence that would suggest that the model based on a reduced data set would yield better fault predictions for the legacy system.

Table 13 Z-test results: *p-values*

CFSS-7 vs. RAW-28		
Data set	AAE	ARE
Release 1	0.2514	0.0000
Release 2	0.1562	0.0465
Release 3	0.0084	0.0004
Release 4	0.0049	0.0007

5.5. Model evaluation for CFSS-7 case study

The correlation-based feature selection algorithm presented in (Hall and Smith, 1998) was used to perform subset selection of the 28 software metrics. The greedy method selected 7 metrics which include: *USAGE*, *CALUNQ*, *FILINCUQ*, *NDSENT*, *NDSEXT*, *NDSPND*, and *VARUSDUQ*. Among the 7 metrics selected by stepwise regression for the RAW-7 case study and the 7 metrics selected for the CFSS-7 case study, there are 4 metrics in common. This implies that these 4 software metrics are important quality indicators for the telecommunications system.

The CBR models were then built using these 7 metrics and the Mahalanobis similarity function and the two solution algorithms: unweighted average and inverse distance weighted average. For each of the two combinations, CBR models were built (using Release 1) by varying the number of nearest neighbors (n_N) and computing the AAE and ARE values. For the CBR model with the unweighted average solution algorithm, $n_N = 27$ based on the AAE performance metric; and $n_N = 33$ based on the ARE performance metric. For the CBR model with the inverse distance unweighted average solution algorithm, $n_N = 27$ for AAE and $n_N = 93$ for ARE. The performance metric values of the CFSS-7 model for the fit and test datasets are shown in Table 11. In general, the RAW-7 CBR model depict better AAE and ARE values than the CFSS-7 CBR model.

Similar to the RAW-7 case study, we compare the performance of the CFSS-7 CBR model with the RAW-28 CBR model using a Z-test. The obtained *p-values*, presented in Table 13, indicate the statistical difference between the two models based on the Mahalanobis similarity measure and the inverse distance weighted average solution algorithm. We compare the two models at a 10% significance level, and use an adjusted *alpha* value for the comparison based on the Bonferroni correction, i.e., $\alpha = 0.0333$.

The table (Table 13) indicates that for AAE, the RAW-28 model performs better (at the reduced α level) than the CFSS-7 model for the Release 3 and Release 4 datasets. A similar observation is made with respect to the ARE performance measure. In the case of the Release 2 dataset, performance improvements of the RAW-28 model with respect to both AAE and ARE are not significant at the reduced α level. Thus, there is no empirical evidence that a CBR model built (for the given system) after performing the correlation-based feature subset selection performs better than a CBR model built without feature subset selection.

5.6. Analysis of variance tests

To investigate whether the difference observed between the respective similarity measures and the respective solution algorithms is of statistical significance, we performed two-way ANOVA tests (Berenson et al., 1983). The similarity function was considered as factor A,

Table 14 ANOVA for release 2: absolute error

	SS	*df*	*MS*	*MS*/*MSE*
A	96.863	2	48.431	150.050
B	14.201	1	14.201	43.999
AB	8.740	3	4.370	13.538
Error	7707.683	23880	0.323	
Total	7827.487	23886		

Table 15 ANOVA for release 2: relative error

	SS	*df*	*MS*	*MS*/*MSE*
A	95.559	2	47.780	160.478
B	12.865	1	12.865	43.208
AB	8.351	3	4.176	14.025
Error	7109.891	23880	0.298	
Total	7226.666	23886		

whereas the solution algorithm was considered as factor B. The term AB indicates whether an interaction exists between the two factors. A log transformation was performed on the absolute error and relative error, because the error values encompassed a wide range and the ratio of the maximum value to the minimum value of the errors was large. The results for Release 2 are shown in Tables 14 and 15, where *SS* is the sum of squares, *df* is the degree of freedom, *MS* is the mean square of factor, and *MSE* is the mean square of error. Results for the other releases are not shown due to similarity in conclusions.

The tables also indicate the significance of an F-test (for the absolute and relative errors) for each factor. A significant effect (large F values) due to factor A is observed. The observed difference among the similarity functions was at a 1% (or lower) significance level for all four releases, suggesting that the selection of a particular (among the three) similarity function has a significant effect on the prediction accuracy. We recall that the prediction performances of models that use the Euclidean or City Block distances are very similar (see Table 9). However, the prediction performances of models that use the Mahalanobis distance are generally better than the models that use the Euclidean or City Block distances. Hence, we can derive (from the ANOVA results) that the Mahalanobis distance performs significantly better than the other two similarity functions for the legacy telecommunications system.

According to the ANOVA modeling results, a significant effect was also observed (very large F values) due to factor B, suggesting that the inverse distance weighted average algorithm provided significantly better results when compared to the unweighted average algorithm. For all four releases the observed improvement was at a 1% (or lower) significance level. Regarding the interaction between the two factors, we observe that in addition to the individual similarity functions and solution algorithms, their combination may also yield significantly different fault predictions.

5.7. Comparison with multiple linear regression

The CBR-based fault prediction approach is compared with the commonly used multiple linear regression (MLR) model (Berenson et al., 1983). The MLR prediction model is built

Table 16 Comparing with multiple linear regression

Modeling method	Release 2		Release 3		Release 4	
	AAE	ARE	AAE	ARE	AAE	ARE
			RAW-28 Case Study			
CBR	0.879	0.528	0.861	0.499	0.831	0.492
MLR	0.890	0.571	0.960	0.602	0.926	0.584
			PCA-10 Case Study			
CBR	0.841	0.539	0.879	0.533	0.807	0.488
MLR	0.875	0.567	0.976	0.626	0.954	0.637

using stepwise regression to select the most significant software metrics at the 5% significance level – 7 metrics were selected. The same selection process was used to extract the seven metrics that formed the datasets for the RAW-7 case study. The MLR model is compared with the CBR model that uses Mahalanobis similarity function and the inverse distance weighted average solution algorithm. The performance metrics of the two models are shown in Table 16.

The table shows the performances for the RAW-28 and PCA-10 case studies for the three test datasets, i.e., Release 2, 3, and 4. It is observed that the CBR models have better performance than the MLR models for both AAE and ARE values. The CBR models have better error values than the MLR models for both case studies. The performance improvement is more pronounced for Release 3 and 4, as compared to Release 2. A Z-test was performed to determine if the CBR models were significantly better than the MLR models for the test datasets. The absolute error and relative error of the models were used as the response variable. The obtained p-values indicated that the CBR models were significantly better (at 5%) than the MLR models for Release 3 and 4. However, for Release 2 the performance improvement was better but not significant.

5.8. Threats to validity

In an empirical software engineering task it is important to consider the threats to validity of the obtained results and conclusions (Wohlin et al., 2000). We consider three types of threats to validity for the case study presented earlier. They include: threats to internal validity, threats to external validity, and threats to conclusion validity.

Threats to internal validity are unaccounted influences that may affect case study results. Internal validity is the ability to show that results obtained were due to the manipulation of the experimental treatment variables. In the context of this study, poor fault prediction can be caused by a wide variety of factors, including measurement errors while collecting and recording software metrics, modeling errors due to the unskilled use of software applications, errors in model-selection during the modeling process, and the presence of noise in the training data set. Measurement errors are inherent to the software data collection effort, which is usually specific to the system under consideration. In our study, a common model-building and model-selection approach have been adopted for each combination of the similarity function and solution algorithm. Moreover, the modeling, experimental, and statistical analysis was performed by only one skilled person in order to keep modeling errors to a minimum.

External validity is concerned with generalization of the obtained results outside the experimental setting. Therefore, threats to external validity are conditions that limit generalization of case study results. To be credible, the software engineering community demands that the subject of an empirical study be a system with the following characteristics (Votta and Porter, 1995): (1) developed by a group, rather than an individual; (2) developed by professionals, rather than students; (3) developed in an industrial environment, rather than an artificial setting; and (4) large enough to be comparable to real industry projects. The software system investigated in this study meets all these requirements.

However, it is known in the software engineering field that a given software metrics-based prediction system is likely to be affected by (among other issues) the characteristics of the data and the application domain of the system under consideration (Shepperd and Kadoda, 2001). The results and conclusion of this paper are based on the case study presented, and cannot be generalized for another software system, because it may: have different software data characteristics; utilize different software metrics; and have different software quality improvement objectives. However, the proposed CBR-based software fault prediction procedure can certainly be applied to other systems.

Conclusion validity is concerned with the statistical relationship between the treatment and the outcome. It is sometimes also referred to as statistical conclusion validity, i.e., is the relationship between the treatment and the outcome satisfiable for a given significance level. Threats to conclusion validity are issues that affect the ability to draw the correct inferences. In this study the discussions related to performance comparisons of the software fault prediction models based on the different combinations of similarity functions and solution algorithms were based on the obtained two-way ANOVA modeling statistics.

6. Conclusion

We have presented the (explorative) use of a computational intelligent technique, i.e., Case-Based Reasoning, for the software fault prediction problem. The simplicity, ease in model-calibration, user-acceptance and prediction accuracy of the CBR quality estimation technique presented demonstrates its practical and applicative attractiveness. Such a modeling system can be used to yield timely fault predictions for software components currently under development, providing useful insights into their quality. The software quality assurance team can then use the predictions to direct the available resources dedicated for reliability improvements, obtaining cost-effective reliability enhancements.

A primary contribution of our study consists of an empirical investigation studying the possible effects of using different similarity functions and solution algorithms on the fault prediction accuracy obtained by our CBR system. In addition, we also explored the model accuracies with respect to a variation in the number of nearest neighbor cases used for determining the response value of the target case. A statistical test using two-way ANOVA models was performed to verify if the results obtained from the different similarity measures and solution algorithms were of any significance.

The lack of a metric selection process in our CBR system resulted in determining the significant metrics prior to CBR analysis. This paper used the stepwise regression model selection technique. In addition to stepwise regression model selection, a greedy selection process for feature subset selection was used to extract significant metrics. Empirical analysis was performed using all the software metrics; using only those selected by stepwise regression; and using only those selected by the greedy selection process.

The software system used for our analysis was a large legacy telecommunications system with four historical software releases. Based on empirical findings of the case study, it is observed that the Mahalanobis distance measure yielded better fault prediction accuracy as compared to the City Block and Euclidean distance measures. However, when the correlation among the software metrics was removed (by principle components analysis), the City Block distance measure demonstrated better accuracy. It should be noted that the removal of correlation among the independent variables involves computational overhead.

It was also observed that among the solution algorithms considered, the inverse distance weighted average depicted a better fault prediction. When compared to prediction models built using multiple linear regression, our CBR-based models generally had much better accuracy. Moreover, case studies of other large-scale software systems provided empirical conclusions similar to as those observed in this paper.

The ANOVA statistical tests based on the original data set, i.e., RAW-28, confirmed the foregoing observations. They also indicated the significant effects due to the interaction between the similarity function and solution algorithm factors. For the telecommunications system investigated in this paper, it was observed that performing a metric-selection procedure prior to using our CBR prediction system, did not yield improved fault predictions. This was observed for both metric subset selection procedures used in our study: stepwise regression model selection and greedy correlation-based feature subset selection.

Hence, in the context of the legacy system case study and our CBR prediction system, the computational overhead involved with performing metric-selection may be avoided. Further empirical evidence in the form of case studies are needed to verify this conclusion. However, we note that software quality estimation with fewer software metrics is more attractive to practitioners from a comprehensibility point of view. In addition, modeling with a small number of software metrics may reduce the software metric data collection efforts for the given software project.

This paper concludes by stating that the CBR system discussed is a promising technology that can be used to obtain a timely and effective software quality prediction. Consequently, a more focused software quality assurance plan can be devised. However, further empirical investigation is warranted to validate our findings.

Future work will include studying the CBR system with case studies of other real-world software systems. An interesting future study would be to discretize the software modules of the dataset into bins and assign different costs of misclassifications to each bin prior to modeling. This would provide a closer insight into the characteristics of the dataset and the performance of the CBR-based software quality model. In addition, a comparative performance evaluation (out of scope for this study) of CBR with respect to other software fault prediction techniques (such as artificial neural networks and regression trees) is needed, and is part of our future work.

Acknowledgments

We are grateful to the anonymous reviewers for their constructive comments and suggestions that helped improve this paper. We thank John P. Hudepohl, Wendell D. Jones and the EMERALD team for collecting the necessary case-study data. We also thank Kehan Gao and Jayanth Rajeevalochanam for their assistance with patient reviews. This work was supported in part by the Cooperative Agreement NCC 2-1141 from NASA Ames Research Center, Software Technology Division, and the NASA Grant NAG 5-12129 for the NASA Software Independent Verification and Validation Facility at Fairmont, West Virginia.

References

Aha, D.W. and Bankert, R.L. 1994. Feature selection for case-based classification of cloud types: an empirical comparison. In D.W. Aha, ed., *Workshop on Case-Based Reasoning (Technical Report WS-94-01)*, Menlo Park, California, AAAI Press.

Bartsch-Spoerl, B. 1995. Toward the integration of case-based, schema-based, and model-based reasoning for supporting complex design tasks. In *Proceedings: First International Conference on Case-Based Reasoning*, pp. 145–156. Springer-Verlag.

Bell, B., Kedar, S. and Bareiss, R. 1994. Interactive model-driven case adaptation for instructional software design. in *Proceedings: 16th Annual Conference of the Cognitive Science Society*, pp. 33–38. Lawrence Erlbaum Publishers.

Berenson, M.L., Levine, D.M. and Goldstein, M. 1983. *Intermediate Statistical Methods and Applications: A Computer Package Approach*. Prentice Hall, Englewood Cliffs, NJ, USA.

Briand, L.C., Langley, T. and Wieczorek, I. 2000. Areplicated assessment and comparison of common software cost modeling techniques. In *Proceedings: International Conference on Software Engineering*, pp. 377–386, Limerick, Ireland. Association for Computing Machinery.

Dillon, W.R. and Goldstein, M. 1984. *Multivariate Analysis: Methods and Applications*. John Wiley & Sons, New York.

Fayyad, U.M. 1996. Data mining and knowledge discovery: making sense out of data. *IEEE Expert*, 11(4): 20–25.

Fenton, N.E. and Pfleeger, S.L. 1997. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing Company: ITP, Boston, MA, 2nd edition.

Ganesan, K., Khoshgoftaar, T.M. and Allen, E.B. 2000. Case-based software quality prediction. *International Journal of Software Engineering and Knowledge Engineering*, 10(2): 139–152. World Scientific Publishing.

Gokhale, S.S. and Lyu, M.R. 1997. Regression tree modeling for the prediction of software quality. In Pham, H., ed. *Proceedings of 3rd International Conference on Reliability and Quality in Design*, pp. 31–36, Anaheim, CA. International Society of Science and Applied Technologies.

Gray, A.R. and MacDonell, S.G. 1999. Software metrics data analysis: exploring the relative performance of some commonly used modeling techniques. *Empirical Software Engineering Journal*, 4: 297–316.

Hall, M.A. and Smith, L.A. 1998. Practical feature subset selection. In *Proceedings: 21st Australian Computer Science Conference*, pp. 181–191. Springer Verlag.

Hudepohl, J.P., Aud, S.J., Khoshgoftaar, T.M., Allen, E.B. and Mayrand, J. 1996. Emerald: Software metrics and models on the desktop. *IEEE Software*, 13(5): 56–60

Idri, A., Abran, A. and Khoshgoftaar, T.M. 2002. Estimating software project effort by analogy based on linguistic values. In *Proceedings: 8th International Software Metrics Symposium*, pp. 21–30, Ottawa, Ontario, Canada, IEEE Computer Society.

Imam, K.E., Benlarbi, S., Goel, N. and Rai, S.N. 2001. Comparing case-based reasoning classifiers for predicting high-risk software componenets. *Journal of Systems and Software*, 55(3): 301–320. Elsevier Science Publishing.

Kadoda, G., Cartwright, M., Chen, L. and Shepperd, M. 2000. Experiences using case-based reasoning to predict software project effort. In *Proceedings of 4th International Conference on Empirical Assessment in Software Engineering*, pp. 23–33, Staffordshire, UK.

Khoshgoftaar, T.M., Allen, E.B. and Busboom, J.C. 2000. Modeling software quality: the software measurement analysis and reliability toolkit. In *Proceedings of 12th International Conference on Tools with Artificial Intelligence*, pp. 54–61, Vancouver, BC, Canada, November. IEEE Computer Society.

Khoshgoftaar, T.M., Bullard, L.A. and Gao, K. 2003. Detecting outliers using rule-based modeling for improving cbr-based software quality classification models. In Ashley, K.D. and Bridge, D.G., (Eds.), *Proceedings of the 16th International Conference on Case-Based Reasoning*, volume 1689, pp. 216–230. Springer-Verlag LNAI.

Khoshgoftaar, T.M., Ganesan, K., Allen, E.B., Ross, F.D., Munikoti, R., Goel, N. and Nandi, A. 1997. Predicting fault-prone modules with case-based reasoning. In *Proceedings of 8th International Symposium on Software Reliability Engineering*, pp. 27–35, Albuquerque, NM, IEEE Computer Society.

Khoshgoftaar, T.M., Nguyen, L., Gao, K. and Rajeevalochanam, J. 2003. Application of an attribute selection method to cbr-based software quality classification. In *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence*, pp. 47–52, Sacramento, CA.

Khoshgoftaar, T.M., Pandya, A.S. and Lanning, D.L. 1995. Application of neural networks for predicting faults. *Annals of Software Engineering*, 1: 141–154.

Khoshgoftaar, T.M. and Seliya, N. 2002 Tree-based software quality models for fault prediction. In *Proceedings: 8th International Software Metrics Symposium*, pp. 203–214, Ottawa, Ontario, Canada IEEE Computer Society.

Khoshgoftaar, T.M. and Seliya, N. 2003. Analogy-based practical classification rules for software quality estimation. *Empirical Software Engineering Journal*, 8(4): 325–350.

Kolodner, J. 1993. *Case-Based Reasoning*. Morgan Kaufmann Publishers, Inc., San Mateo, California USA.

Korel, B. 1996. Automated test data generation for programs with procedures. *Proceedings of the International Symposium on Software Testing and Analysis*, 21(3): 209–215.

Kriegsman, M. and Barletta, R. 1993. Building a case-based help desk application. *IEEE Expert*, 8(6): 18–24.

Leake, D.B. 1996. Editor. *Case-Based Reasoning: Experience, Lessons and Future Directions*. MIT Press, Cambridge, MA USA.

Perry, W.E. 2000. *Effective Methods for Software Testing*. John Wiley & Sons, New York, NY, 2nd edition.

Porter, A.A., Siy, H.P., Toman, C.A. and Votta, L.G. 1997. An experiment to assess the cost-benefits of code-inspection in large scale software development. *IEEE Transactions on Software Engineering*, 23(6): 329–346.

Ramamoorthy, C.V., Chandra, C., Ishihara, S. and Ng, Y. 1993. Knowledge-based tools for risk assessment in software development and reuse. In *Proceedings: 5th International Conference on Tools with Artificial Intelligence*, pp. 364–371, Boston, MA, USA IEEE Computer Society.

Schneidewind, N.F. 2002. Body of knowledge for software quality measurement. *IEEE Computer*, 35(2): 77–83

Shepperd, M. and Kadoda, G. 2001. Comparing software prediction techniques using simulation. *IEEE Transactions on Software Engineering*, 27(11): 1014–1022.

Shepperd, M. and Schofield, C. 1997. Estimating software project effort using analogies. *IEEE Transactions on Software Engineering*, 23(12): 736–743.

Smith, N.T. and Ganesan, K. 1995. Software design using case-based reasoning. In *Proceedings: Fourth Software Engineering Research Forum*, pp. 193–200, Boca Raton, FL

Sundaresh, N. 2001. An empirical study of analogy based software fault prediction. *Master's thesis, Florida Atlantic University*, Boca Raton, FL. Advised by Taghi M. Khoshgoftaar.

Troster, J. and Tian, J. 1995. Measurement and defect modeling for a legacy software system. *Annals of Software Engineering*, 1: 95–118

Votta, L.G. and Porter, A.A. 1995. Experimental software engineering: a report on the state of the art. In *Proceedings of the 17th. International Conference on Software Engineering*, pp. 277–279, Seattle, WA USA. IEEE Computer Society.

Whitten, I.H. and Frank, E. 2000. *Data Mining: Practical Machine Learning Tools and Techniques with JAVA Implementations*. Morgan Kaufmann, San Francisco, CA.

Wohlin, C., Runeson, P., Host, M., Ohlsson, M.C., Regnell, B. and Wesslen, A. 2000. *Experimentation in Software Engineering: An Introduction*. Kluwer International Series in Software Engineering. Kluwer Academic Publishers, Boston, MA.

Taghi M. Khoshgoftaar is a professor of the Department of Computer Science and Engineering, Florida Atlantic University and the Director of the Empirical Software Engineering Laboratory. His research interests are in software engineering, software metrics, software reliability and quality engineering, computational intelligence, computer performance evaluation, data mining, and statistical modeling. He has published more than 200 refereed papers in these areas. He has been a principal investigator and project leader in a number of projects with industry, government, and other research-sponsoring agencies. He is a member of the Association for Computing Machinery, the IEEE Computer Society, and IEEE Reliability Society. He served as the general chair of the 1999 International Symposium on Software Reliability Engineering (ISSRE'99), and the general chair of the 2001 International Conference on Engineering of Computer Based Systems. Also, he has served on technical program committees of various international conferences, symposia, and workshops. He has served as North American editor of the Software Quality Journal, and is on the editorial boards of the journals Empirical Software Engineering, Software Quality, and Fuzzy Systems.

Naeem Seliya received the M.S. degree in Computer Science from Florida Atlantic University, Boca Raton, FL, USA, in 2001. He is currently a Ph.D. candidate in the Department of Computer Science and Engineering at Florida Atlantic University. His research interests include software engineering, computational intelligence, data mining, software measurement, software reliability and quality engineering, software architecture, computer data security, and network intrusion detection. He is a student member of the IEEE Computer Society and the Association for Computing Machinery.