
Palwinder Kaur Bhangu, Manveet Kaur, Shikha Verma, Sukhwinder Kaur

Abstract—In this study, a survey on prediction of fault proneness in software systems is performed. The work of various researchers is discussed in brief.

Keywords—Software, Metrics, Function Based, Object Oriented, Fault.

I. INTRODUCTION

Bellini in his paper titled “Comparing Fault-Proneness Estimation Models” compared Fault-Proneness Estimation Models and concluded that over the last years, software quality has become one of the most important requirements in the development of systems and fault-proneness estimation could play a key role in quality control of software products [1]. Their main objective was to find a compromise between the fault-proneness estimation rate and the size of the estimation model in terms of number of metrics used in the model itself. The methodologies used in their study were logistic regression and discriminate analysis.

Challagulla in his paper “A Unified framework for Defect Data Analysis using the MBR Technique” suggested the use of MBR classifier to effectively estimate defects from prior data [2]. Better prediction models facilitate better project planning and risk/cost estimation to reduce the V&V cost for achieving high confidence levels. A framework was developed using MBR classifier for software defect data by logical variations of its configuration parameters. Accuracy was predicted using different parameters such as true positives, false positives, true negatives, and false negatives. Finally, they concluded that if accuracy is the only criteria to estimate the software quality, then MBR with Euclidean distance and one nearest neighbor comes out be best prediction method.

Costa and Oliveira in their paper on “Cluster Analysis using Growing Neural Gas and Graph Partitioning” discussed about the size and complexity of data sets [3]. The size and complexity of data sets is ever increasing. Clustering, considered the most important unsupervised learning problem, is used to reveal structures and identify “natural” groupings on the multivariate data. Several competitive learning algorithms were developed for his application. The Growing Neural Gas (GNG) is an incremental algorithm, where no previous information about the number of clusters is preset. New units are added according the training dynamics. GNG produces a graph that represents the topology of trained data. Each vertex corresponds to a neuron in which input data have been mapped. This paper describes a simple algorithm to better produce the partitioning of this graph, generating connected components that represent different data clusters. The algorithm automatically finds the number of classes and the associated neurons.

Emam et al. in his paper “Comparing case based reasoning classifiers for predicting high-risk software components” proposed Case-based reasoning (CBR) for predicting the risk class of software components [5]. Risky components can be defined as those that are fault-prone, or those that require a large amount of effort to maintain. Thus far evaluative studies of CBR classifiers have been promising, showing that their predictive performance is as good as or better than other types of classifiers. However, a CBR classifier can be instantiated in different ways by varying its parameters, and it is not clear which combination of parameters provides the best performance. In this paper we evaluate the performance of a CBR classifier with different parameters, namely: (a) different distance measures, (b) different standardization techniques, (c) use or non-use of weights, and (d) the number of nearest neighbors to use for the prediction. In total, we compared 30 different CBR classifiers. The study was conducted with a data set from a large real-time system, and the objective was to predict the fault-proneness of its components. Our results indicate that there is no difference in prediction performance when using any combination of parameters. Based on these results, we recommend using a simple CBR classifier with Euclidean distance, z-score standardization, no weighting scheme, and selecting the single nearest neighbor for prediction. The advantage of such a classifier is its intuitive appeal to nonspecialists, and the fact that it performs as well as more complex classifiers.

Fenton in his paper titled “A Critique of Software Defect Prediction Models” provided a critical review of the literature and made heroic contributions to the subject otherwise bereft of empirical studies [9]. In their study, most of the wide range of prediction models used size and complexity metrics to predict defects. Others are based on testing data, the quality of the development process, or take a multivariate approach. They proposed that the models are weak because of their inability to cope with the, as yet, unknown relationship...
between defects and failures. There are fundamental statistical and data quality problems that undermine model validity. More significantly many prediction models tend to model only part of the underlying problem and seriously ill-specify it. They recommended holistic models for software defect prediction, using Bayesian Belief Networks, as alternative approaches to the single-issue models used. They also argued for research into a theory of software decomposition in order to test hypotheses about defect introduction and help construct a better science of software engineering.

Gillies published a book titled “Software quality”. An extract from the book “4.1: The work of Gilb” discussed the work of Gilb with the models of McCall et al. (1977) and Boehm (1978) [10]. There are two significant strands to the work. The first strand addresses the development process itself. In addition to the evolutionary delivery method is the use of quality template, rather than a rigid hierarchical model. The key feature of the template is that it is designed to be tailored to the local requirements. The philosophy behind this is that quality depends principally upon a small set of critical resources, which will vary from one application to another. Within such a view, the role of software engineering is to identify which quality criteria are critical and define the extent to which these must be present. Evolutionary developments are seen as critical by Gilb (1988) to the satisfaction of these critical criteria. It is an iterative approach aiming to converge towards clear and measurable multidimensional objectives. At each stage, the developer intends to maximize the distance moved towards the ultimate objectives whilst minimizing resource expended. Gilb proposes four quality attributes: workability, availability, adaptability and usability, accompanied by the resource attributes of time, money, people and tools. However, the work has been criticized because the template is defined for each application, precluding comparison and making quality measurements very time and resource consuming.

Giovanni in his paper “Estimating software fault-proneness for tuning testing activities” suggested that there is an existence of a correlation between a reasonable set of static metrics and software fault-proneness [11]. Static metrics, e.g., the McCabe's cyclomatic number or the Halstead's Software Science, statically computed on the source code, try to quantify software complexity. Dynamic metrics, e.g., structural and data flow coverage measure the thoroughness of testing as the amount of elements of the program covered by test executions. Such metrics only partially reflect the many aspects that influence the software fault-proneness, and thus provide limited support for tuning the testing process.

Guo in his paper titled “Predicting fault prone modules by the Dempster–Shafer belief networks” describes a novel methodology for predicting fault prone modules [13]. The methodology is based on Dempster-Shafer (D-S) belief networks. Our approach consists of three steps: First, building the Dempster-Shafer network by the induction algorithm. Second, selecting the predictors (attributes) by the logistic procedure. Third, feeding the predictors describing the modules of the current project into the inducted Dempster-Shafer network and identifying fault prone modules. We applied this methodology to a NASA dataset. The prediction accuracy of our methodology is higher than that achieved by logistic regression or discriminant analysis on the same dataset.

Jiang in his paper “Fault Prediction Using Early Lifecycle Data” presented the application of machine learning algorithms in software quality estimation using metrics available early in the development lifecycle [15]. Their study was carried out using software measurement data of three NASA projects JM1, CM1 and PC1. In their paper, they compared the performance of requirement based models against the performance of code-based models and models that combine requirement and code metrics. They used these prediction models for prediction of fault-prone modules in a software project. Their study indicates that early lifecycle metrics can play an important role in project management either by pointing to the need for increased quality monitoring during the development or by using the models to assign verification and validation activities. They concluded that combing metrics that describe different yet related software artifacts may significantly increase the effectiveness of defect prediction models.

Khoshgaftaar and Seliya in his paper “Software quality classification modeling using the SPRINT decision tree algorithm” predicting the quality of system modules prior to software testing and operations can benefit the software development team [17]. Such timely reliability estimation can be used to direct cost-effective quality improvement efforts to the high-risk modules. Tree-based software quality classification models based on software metrics are used to predict whether a software module is fault-prone or not fault-prone. They are white box quality estimation models with good accuracy, and are simple and easy to interpret. This paper presents an in-depth study of calibrating classification trees for software quality estimation using the SPRINT decision tree algorithm. Many classification algorithms have memory limitations including the requirement that data sets be memory resident. SPRINT removes all of these limitations and provides a fast and scalable analysis. It is an extension of a commonly used decision tree algorithm, CART, and provides a unique tree-pruning technique based on the Minimum Description Length (MDL) principle. Combining the MDL pruning technique and the modified classification algorithm, SPRINT yields classification trees with useful prediction accuracy. The case study used comprises of software metrics and fault data collected over four releases from a very large telecommunications system. It is observed that classification trees built by SPRINT are more balanced and demonstrate better stability in comparison to those built by CART.

Khoshgoftaar and Seliya in their paper “Analogy-based practical classification rules for software quality estimation” suggested that the Software metrics-based quality estimation models can be effective tools for identifying which modules
are likely to be fault-prone or not fault-prone [18]. The use of such models prior to system deployment can considerably reduce the likelihood of faults discovered during operations, hence improving system reliability. A software quality classification model is calibrated using metrics from a past release or similar project, and is then applied to modules currently under development. Subsequently, a timely prediction of which modules are likely to have faults can be obtained. However, software quality classification models used in practice may not provide a useful balance between the two misclassification rates, especially when there are very few faulty modules in the system being modeled. In this paper they presented, in the context of case-based reasoning, two practical classification rules that allow appropriate emphasis on each type of misclassification as per the project requirements. The suggested techniques are especially useful for high-assurance systems where faulty modules are rare. The proposed generalized classification methods emphasize on the costs of misclassifications, and the unbalanced distribution of the faulty program modules. We illustrate the proposed techniques with a case study that consists of software measurements and fault data collected over multiple releases of a large-scale legacy telecommunication system. In addition to investigating the two classification methods, a brief relative comparison of the techniques is also presented. It is indicated that the level of classification accuracy and model-robustness observed for the case study would be beneficial in achieving high software reliability of its subsequent system releases. Similar observations are made from our empirical studies with other case studies.

Khoshgaftaar and Seliya in his paper “Comparative assessment of Software quality classification techniques” explained Software metrics-based quality classification models that predict a software module as either fault-prone (fp) or not fault-prone (nfp) [19]. Timely application of such models can assist in directing quality improvement efforts to modules that are likely to be fp during operations, thereby cost-effectively utilizing the software quality testing and enhancement resources. Since several classification techniques are available, a relative comparative study of some commonly used classification techniques can be useful to practitioners. We present a comprehensive evaluation of the relative performances of seven classification techniques and/or tools. These include logistic regression, case-based reasoning, classification and regression trees (CART), tree-based classification with S-PLUS, and the Sprint-Sliq, C4.5, and Treedisc algorithms. The use of expected cost of misclassification (ECM), is introduced as a singular unified measure to compare the performances of different software quality classification models. A function of the costs of the Type I (anfp module misclassified as fp) and Type II (a fp module misclassified as nfp) misclassifications, ECM is computed for different cost ratios. Evaluating software quality classification models in the presence of varying cost ratios is important, because the usefulness of a model is dependent on the system-specific costs of misclassifications. Moreover, models should be compared and preferred for cost ratios that fall within the range of interest for the given system and project domain. Software metrics were collected from four successive releases of a large legacy telecommunications system. A two-way ANOVA randomized-complete block design modeling approach is used, in which the system release is treated as a block, while the modeling method is treated as a factor. It is observed that predictive performances of the models is significantly different across the system releases, implying that in the software engineering domain prediction models are influenced by the characteristics of the data and the system being modeled. Multiple-pairwise comparisons are performed to evaluate the relative performances of the seven models for the cost ratios of interest to the case study. In addition, the performance of the seven classification techniques is also compared with a classification based on lines of code. The comparative approach presented in this paper can also be applied to other software systems.

Lanubile in his paper “Comparing Models for Identifying Fault-Prone Software Components” presented an empirical investigation of the modeling techniques for identifying fault-prone software components early in the software life cycle [21]. Using software complexity measures, the techniques build models, which classify components as likely to contain faults or not. The modeling techniques applied in their study cover the main classification paradigms, including principal component analysis, discriminate analysis, logistic regression, logical classification models and layered neural networks. In this paper he discussed all the modeling techniques used for classification of modules as faulty or not. He suggested that no model was able to discriminate between components with faults and components without faults.

Munson and Khoshgoftaar in their paper titled “The Detection of Fault-Prone Programs” suggested that the use of statistical technique of discriminant analysis as a tool for the detection of fault-prone programs is explored [23]. A principal-components procedure was employed to reduce simple multicollinear complexity metrics to uncorrelated measures on orthogonal complexity domains. These uncorrelated measures were then used to classify programs to alternate groups depending on the metric value of the program. The criterion variable for group determination was a quality measure of faults or changes made to the programs. The dicriminant analysis was conducted on two distinct data sets from large commercial systems. The basic discriminant model was constructed from deliberately biased data to magnify difference in metric values between the discriminant groups. The technique was successful in classifying programs with a relatively low error rate. While the use of linear regression models has produced models of limited value, this procedure shows great promise for use in the detection of program modules of with high potential for faults.

Schneidewind in their paper titled “Investigation of logistic regression as a discriminant of software quality”
investigated the possibility that Logistic Regression Functions (LRFs), when used in combination with Boolean Discriminant Functions (BDFs), which we had previously developed, would improve the quality classification ability of BDFs when used alone [26]. This was the case; when the union of a BDF and LRF was used to classify quality, the predicative accuracy of quality and inspection cost was improved over that of using either function alone for the Space Shuttle. Also, the LRFs proved useful for ranking the quality of modules in a build. The significance of these results is that very high quality classification accuracy (1.25% error) can be obtained while reducing the inspection cost incurred in achieving high quality. This is particularly important for safety critical systems. Because the methods are general and not particular to the Shuttle, they could be applied to other domains. A key part of the LRF development was a method for identifying the critical Shuttle, they could be applied to other domains. A key part of the LRF development was a method for identifying the critical

Shuttle, they could be applied to other domains. A key part of the LRF development was a method for identifying the critical Shuttles, they could be applied to other domains. A key part of the LRF development was a method for identifying the critical Shuttles, they could be applied to other domains. A key part of the LRF development was a method for identifying the critical Shuttles, they could be applied to other domains. A key part of the LRF development was a method for identifying the critical Shuttles, they could be applied to other domains. A key part of the LRF development was a method for identifying the critical Shuttles, they could be applied to other domains. A key part of the LRF development was a method for identifying the critical Shuttles, they could be applied to other domains. A key part of the LRF development was a method for identifying the critical Shuttles, they could be applied to other domains. A key part of the LRF development was a method for identifying the critical

Seliya and Khoshgoftaar in his paper “Software Quality Analysis of Unlabeled Program Modules with Semi supervised Clustering” estimated the quality of software using semi-supervised clustering technique [24]. They used k-means clustering on the datasets obtained from multiple national Aeronautics and Space administration software projects. They proposed that semi-supervised clustering is useful even if there is no knowledge of software fault proneness data. Finally they concluded that there are some problem areas due to errors during software data collection, problems with data collection tools, issues inherent to specific software processes adopted during development. However, there is need for best prediction model that can even predict these noisy instances.

Yau in his paper “An Application of Fuzzy Clustering to Software Quality Prediction” suggested that predicted the faulty/non-faulty modules by presenting a modeling technique that integrates fuzzy clustering with module order modeling for software quality prediction [27]. They conducted a case study to predict whether module will be considered fault prone or not. Their case study found that one can classify modules which will have faults which will discover by customers with useful accuracy prior to release.

Zhong in his paper “Analyzing software measurement data with clustering techniques” explained the various clustering techniques for the evaluation of software components [28]. For software quality estimation, software development practitioners typically construct quality-classification or fault prediction models using software metrics and fault data from a previous system release or a similar software project. Engineers then use these models to predict the fault proneness of software modules in development. Software quality estimation using supervised-learning approaches is difficult without software fault measurement data from similar projects or earlier system releases. Cluster analysis with expert input is a viable unsupervised-learning solution for predicting software modules’ fault proneness and potential noisy modules. These models are used by engineers to predict the fault proneness. Data analysts and software engineering experts can collaborate more closely to construct and collect more informative software metrics.

REFERENCES


