Expert Systems with Applications 38 (2011) 4626–4636

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Review

Software fault prediction: A literature review and current trends

Cagatay Catal

The Scientific and Technological Research Council of Turkey (TUBITAK), Marmara Research Center, Information Technologies Institute, Kocaeli, Turkey

ARTICLE INFO

Keywords:
Machine learning
Automated fault prediction models
Expert systems
Software quality engineering
Software engineering
Statistical methods

ABSTRACT

Software engineering discipline contains several prediction approaches such as test effort prediction, correction cost prediction, fault prediction, reusability prediction, security prediction, effort prediction, and quality prediction. However, most of these prediction approaches are still in preliminary phase and more research should be conducted to reach robust models. Software fault prediction is the most popular research area in these prediction approaches and recently several research centers started new projects on this area. In this study, we investigated 90 software fault prediction papers published between year 1990 and year 2009 and then we categorized these papers according to the publication year. This paper surveys the software engineering literature on software fault prediction and both machine learning based and statistical based approaches are included in this survey. Papers explained in this article reflect the outline of what was published so far, but naturally this is not a complete review of all the papers published so far. This paper will help researchers to investigate the previous studies from metrics, methods, datasets, performance evaluation metrics, and experimental results perspectives in an easy and effective manner. Furthermore, current trends are introduced and discussed.

1. Introduction

Software fault prediction approaches use previous software metrics and fault data to predict fault-prone modules for the next release of software. If an error is reported during system tests or from field tests, that module's fault data is marked as 1, otherwise 0. For prediction modeling, software metrics are used as *independent variables* and fault data is used as the *dependent variable*. Therefore, we need a version control system such as Subversion to store source code, a change management system such as Bugzilla to record faults, and a tool to collect product metrics from version control system. Parameters of the prediction model are computed by using previous software metrics and fault data.

Software fault prediction models have been studied since 1990s until now and fault-prone modules can be identified prior to system tests by using these models. According to recent studies, the probability of detection (PD) (71%) of fault prediction models may be higher than PD of software reviews (60%) if a robust model is built (Menzies, Greenwald, & Frank, 2007a).

A panel at IEEE Metrics 2002 conference did not support Fagan's claim that inspections can find 95% of defects before testing and this detection ratio was around 60% (Menzies et al., 2007a). One member of the review team may inspect 8–20 lines of code per minute and this process is performed by all the members of a team consisting of 4–6 members (Menzies et al., 2007a). Therefore, software fault prediction approaches are much more cost-effective to detect software faults compared to software reviews.

Benefits of software fault prediction are listed as follows (Catal & Diri, 2009a):

- Reaching a highly dependable system,
- Improving test process by focusing on fault-prone modules,
- Selection of best design from design alternatives using object-oriented metrics,
- Identifying refactoring candidates that are predicted as fault-prone,
- Improving quality by improving test process.

In this study, we investigated 90 software fault prediction papers published between year 1990 and 2009. We categorized these papers according to the publication year and investigated them from numerous perspectives. This paper surveys the software engineering literature on software fault prediction and both machine learning based and statistical based approaches are included in this survey. Papers explained in this article reflect the outline of what was published so far, but naturally this is not a complete review of all the papers. We excluded position papers which do not include experimental results. The inclusion of papers was based on the similarity degree of the study with fault prediction. The exclusion did not take into account the publication year of the paper or methods which have been used.

This paper is organized as follows: Section 2 explains papers which have been investigated in this study. Section 3 provides

E-mail addresses: cagatay.catal@bte.mam.gov.tr, cagatay.catal@hotmail.com

0957-4174/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2010.10.024

the practical software fault prediction problems and shows significant researchers in this area. Section 4 presents the conclusions and suggestions.

2. Papers in review

In the following subsections, we explain papers investigated in this study according to their publication year. Papers in review are conference proceedings, journal articles, and book chapters. A categorization based on publication year reflects the change in this research area between year 1990 and year 2009. Total number of papers in review is 90 and distribution of them according to years is shown as follows:

- Year 1990–Year 2000: **10** papers.
- Year 2000–Year 2003: **14** papers.
- Year 2003–Year 2005: **15** papers.
- Year 2005–Year 2007: **17** papers.
- Year 2007–Year 2009: **34** papers.

2.1. Studies between Year 1990 and Year 2000 (10 papers in review)

Porter and Selby (1990) used classification trees with method level metrics on two software systems of NASA and Hughes Aircraft and the model they built had 79.3% accuracy. However, accuracy metric is not appropriate for software fault prediction studies because imbalanced datasets cannot be evaluated with this metric.

Briand, Basili, and Hetmanski (1993) applied logistic regression, classification trees, optimized set reduction (OSR) methods on 146 components of an ADA system which consists of 260,000 lines of code. OSR method was developed during TAME project in University of Maryland and it was investigated to solve several software engineering problems such as project cost estimation. They chose *correctness* and *completeness* performance metrics to evaluate their prediction models. The best performance was achieved with OSR technique in this study and correctness/completeness was 90%.

Lanubile, Lonigro, and Visaggio (1995) compared models based on principal component analysis, discriminant analysis, logistic regression, logical classification approaches, layered neural networks, and holographic networks for identifying fault-prone components on 27 academic projects developed in University of Bari. They used 11 method level metrics which include Halstead, McCabe, and Henry and Kafura information flow metrics. The evaluation was performed based on misclassification rate, achieved quality, and verification cost. None of these approaches provided acceptable results and they could not make a distinction between fault-prone and non fault-prone modules.

Cohen and Devanbu (1997) used FOIL and FLIPPER methods based on Inductive Logic Programming (IPL) and applied class level coupling metrics on a Management Information System developed in University of Maryland. Evaluation metrics were error rate and time. FLIPPER was faster than FOIL and its performance was better than FOIL. When datasets had noisy data, FLIPPER's performance was better compared to FOIL and it was faster on artificial datasets, too. They suggested two modifications on FLIPPER at the end of this study.

Khoshgoftaar, Allen, Hudepohl, and Aud (1997) applied artificial neural networks and discriminant model on a very large telecommunications system which consists of 13 million lines of code and PROTEL language was used to develop this system. Type-I, Type-II, and overall misclassification rates were used as performance evaluation metrics. In this study, one module was defined as a set of source code files which work together to perform a function. Artificial neural networks' performance was better than discriminant model and this software was integrated to EMERALD software.

Evett, Khoshgoftar, Chien, and Allen (1998) predicted quality based on genetic programming on a military communication system and telecommunications system. They used eight method level metrics which consist of McCabe, Halstead metrics and lines of code metric. *Ordinal evaluation procedure* was the performance evaluation approach and genetic programming was first used in quality prediction in this study. Performance was quite well on these datasets.

Ohlsson, Zhao, and Helander (1998) used principal component analysis for feature selection and then applied discriminant analysis for classification. Dataset was from Ericsson telecommunications system and design metrics, collected from design documents, were independent variables. Design metrics were easily collected from design documents because Ericsson designers applied Formal Definition Language – FDL during design phase. Misclassification rate was the performance evaluation metric and discriminant analysis did no provide good results on Ericsson dataset. They investigated the statistical methodology in this method and stated that the problem was the estimation of probability density function. They improved this approach by taking care of this estimation and they had good results with *multivariate analysis* techniques. When they used more than two groups (faulty and non-faulty) for modules, the performance improved.

Binkley and Schach (1998) predicted run-time failures on OASIS course registration system by using Spearman rank correlation test and static product quality metrics. These metrics are inter-module metrics such as fan-in and intra-module metrics such as cyclomatic complexity. They reported that it is possible to predict the run-time behavior of systems with static inter-module metrics. They also stated that there is a correlation between these inter-module metrics and run-time failures.

De Almeida and Matwin (1999) studied on software quality modeling on Bell Canada's programs by using method level metrics such as Halstead, McCabe and lines of code metrics. They applied C4.5, NewID, CN2, and FOIL methods and used correctness, completeness and accuracy parameters as evaluation metrics. Performance of FOIL was worse than the other algorithms and NewID, C4.5, and CN2 provided similar performance results. From understandability perspective, C4.5's rules were much appropriate for this domain. They stated that statistical approaches work like black-box and they are highly dependent to dataset. Also, they reported that using data from other organizations is not suitable for software quality modeling.

Kaszycki (1999) used process metrics such as programmer experience in addition to the product metrics in order to enhance software fault prediction models. Four product metrics were extracted from source code. True Negative and True Positive rates were used as evaluation metrics. They stated that the performance of models improved when process metrics were included in the metric set and process metrics were integrated into EMERALD product which is a risk assessment tool developed in Nortel Networks. This product builds models based on complexity and process metrics. The importance of this study is the usage of process metrics in the context of models. However, process metrics can change when companies reorganize and processes change. This means that models should be built from scratch each time the process or organization structure changes.

2.2. Studies between Year 2000 and Year 2003 (14 papers in review)

Yuan, Khoshgoftaar, Allen, and Ganesan (2000) applied *fuzzy subtractive clustering* method to predict the number of faults and then, they used *module-order modeling* to classify the modules into faulty/non-faulty classes. They developed four models with 10 method-level metrics and in addition to these four models; they built a model which utilizes from process and product metrics.

Type-I error, Type-II error and overall misclassification rate, effectiveness, and efficiency were used as performance evaluation metrics. They stated that process metrics do not improve the classification accuracy and such a model does not provide acceptable results. In this study, only faults reported by customers were used and one module was defined as a set of source code files which work together to perform a function.

Denaro (2000) estimated software fault-proneness of modules in antenna configuration software by using logistic regression and 37 metrics. R^2 was the evaluation parameter in this study. He showed that there is a correlation between static metrics and fault-proneness.

Khoshgoftaar, Allen, and Busboom (2000) predicted software quality by using case-based reasoning and eight method level metrics for a command-control-communication system developed with ADA language. Method level metrics are lines of code, Halstead, and McCabe metrics. Type-I and Type-II error were used as performance evaluation metrics and prediction model was very useful.

Xu, Khoshgoftaar, and Allen (2000) first used principal component analysis for feature selection and then applied fuzzy nonlinear regression (FNR) to predict faults on a large telecommunications system developed with Protel language. They used 24 method level metrics and four execution metrics which can be collected with EMERALD tool. Performance evaluation metric was average absolute error. They reported that fuzzy nonlinear regression method is an encouraging technology for early fault prediction. FNR modeling produces fuzzy values and predicts an interval for each module. This interval shows the number of faults for the next release. In this study, a module was defined as a set of source code files.

Guo and Lyu (2000) applied *finite mixture model analysis* with Expectation–Maximization (EM) algorithm to predict software quality on a medical imaging system developed with Pascal and Fortran languages. Method level metrics such as Halstead, McCabe, Belady's bandwidth, Jensen's program length were used and Type-I, Type-II, and overall misclassification rates were used as performance evaluation metrics. This study showed that this approach can predict the quality even if fault data are not complete. Also, the best value for Type-II error was 13% and this is similar to the value produced with discriminant analysis.

Khoshgoftaar, Gao, and Szabo (2001) predicted faults by using *zero-inflated poisson regression model* (ZIP) and file level metrics on two large scale software applications. File metrics were the number of source code inspections before system tests, the length of source code prior to system tests, the length of comments in source code prior to coding (auto-generated code), and the length of comments in source code prior to system tests. Performance evaluation metrics were average absolute error (AAE) and average relative error (ARE). In this study, poisson regression model (PRM) and ZIP models were investigated. Hypothesis test was performed by using Vuong test. This study showed that ZIP provided better results compared to PRM. In addition, AAE and ARE values of ZIP was smaller than PRM's error results.

Schneidewind (2001) investigated Boolean discriminant functions (BDF) and logistic regression functions (LRF) for software quality prediction by using six method level metrics on a spacecraft software dataset. Type-I error, Type-II error, overall misclassification rate, and the rate of correctly classified non-faulty modules (LQC) parameters were used as performance evaluation metrics. BDF's performance was better than LRF's performance and the inspection cost increased when LRF and BDF methods were combined.

Emam, Melo, and Machado (2001) predicted the fault-prone classes by using logistic regression and class level metrics on a commercial Java application. Two metrics from Chidamber–Kemerer metrics suite and eight metrics from Briand's metrics suite were selected. In addition to these metrics, size metric was used. J coefficient was the performance evaluation metric in this study. They reported that inheritance depth and export coupling are the most useful metrics to identify the fault-prone classes. In their previous study, they had stated that inheritance depth is not significant. The design quality of this software can be low and this may affect the research results. According to this study, export coupling metric is more useful than inheritance depth metric.

Khoshgoftaar, Geleyn, and Gao (2002) used PRM, ZIP, and module-order modeling techniques with five file level metrics on two applications which configure wireless products. Average absolute error and average relative error were performance evaluation metrics. The performance of module-order model was quite well on these datasets and other approaches did not provide good results.

Khoshgoftaar (2002) applied *Generalized Boolean Discriminant Functions* (GBDF) technique for software quality classification on a military command-control-communication system by using eight method level metrics (Halstead and McCabe). Type-I error and Type-II error parameters were used as performance evaluation metrics. Khoshgoftaar reported that GBDF provided better results compared to BDF. Type-I error of BDF was very high and inspection cost of BDF model was high.

Mahaweerawat, Sophasathit, and Lursinsap (2002) first used fuzzy clustering and then, they applied radial basis function (RBF) to predict software faults. Type-I error, Type-II error, overall misclassification rate, inspection, and completeness were performance evaluation metrics. Method level metrics were Halstead, McCabe, and Henry and Kafura metrics. RBF provided better results than multi layer perceptron. Even though MLP's completeness value was higher, its inspection cost was high too. RBF's accuracy was 83% and MLP's accuracy was 60%. Therefore, RBF method was better than MLP for software fault prediction in this study.

Khoshgoftaar and Seliya (2002a) applied SPRINT and CART methods with 28 method level metrics (24 product metrics and four execution metrics) on a large telecommunications system for software quality classification. SPRINT is a classification tree algorithm and CART is decision tree algorithm. Performance evaluation metrics were Type-I error, Type-II error and overall misclassification rate. They reported that SPRINT algorithm had lower Type-I error and the model based on SPRINT was robust. However, the tree structure of SPRINT was more complex than the tree of CART.

Pizzi, Summers, and Pedrycz (2002) predicted software quality by using multi layer perceptron and method/class level metrics on a research prototype. Accuracy parameter was used as performance evaluation metric. They stated that using median-adjusted class labels is an effective pre-processing technique before multi layer perceptron is applied.

Khoshgoftaar and Seliya (2002b) used tree based software quality prediction models on a large telecommunications system and they applied design metrics. CART-LS (least squares), S-PLUS, and CART-LAD (least absolute deviations) were investigated in this study. Evaluation parameters were average absolute error and average relative error. They reported that CART-LAD's accuracy was quite well and its interpretation was simple. However, S-PLUS's performance was not well and the classification tree was very complex. Therefore, CART-LAD was proposed for software quality prediction.

2.3. Studies between Year 2003 and Year 2005 (15 papers in review)

Reformat (2003) applied fuzzy rule-based models for reasoning about the number of software faults and he used 11 method level metrics on a commercial medical imaging system. He stated that nine classification models' classification rates change between

62.82% and 79.49%. Classification rate was 85.90% when Meta model prediction system was used.

Koru and Tian (2003) investigated the relationship between high defect and high complexity modules by using tree based models and method level metrics (15 method level metrics for IBM products and 49 method level metrics for Nortel Networks) on six large scale products of IBM and Nortel Networks. Mann–Whitney *U*-test was applied for performance evaluation. They showed that high defect-prone modules are generally complex modules, but they are not the most complex ones. They also reported that the highest defect-prone modules are the modules which locate below the most complex modules. According to some researchers from IBM and Nortel Networks, inherent complexity of the problem causes some modules to be very complex.

Denaro, Lavazza, and Pezzè (2003) applied logistic regression by using class level metrics on an industrial telecommunications system for fault prediction. CK metrics except coupling between object classes (CBO) and lack of cohesion of methods (LCOM) were used in this study because CBO and LCOM metrics could not be calculated with automated tools. They stated that none of these metrics is correlated with fault-proneness and multivariate models do not provide any advantage compared to lines of code metric. Software used in this study was first implemented with C language and later this software evolved to an object oriented system. This evolution of system from C to object oriented system and a bad object oriented design may be the major causes of correlation problem stated above. Normally researchers expected to see a correlation between fault-proneness and at least one metric. A module was the combination of a header and its corresponding cpp file.

Thwin and Quah (2003) predicted software quality based on General Regression Neural Network (GRNN) and Ward Neural Networks. They used class level metrics and reported that GRNN provided much better results than Ward networks. Evaluation parameters were R^2, *r*, average square error, average absolute error, minimum absolute error, and maximum absolute error.

Khoshgoftaar and Seliya (2003) applied CART-LS, CART-LAD, S-PLUS, multiple linear regression, neural networks, case based reasoning on a large telecommunications system. Twenty-four product and four execution metrics were independent variables for this analysis. They used two-way ANOVA randomized-complete block design model as experimental design approach and multiple-pairwise comparison for performance ranking. Best performance was achieved with CART-LAD algorithm and the worst one was S-PLUS. Furthermore, they stated that principal component analysis does not always necessarily improve the performance of models, but it removes the correlation between metrics and the model becomes more robust.

Guo, Cukic, and Singh (2003) predicted fault-prone modules on NASA's KC2 project by using Dempster–Shafer Belief Networks and 21 method level metrics. Performance evaluation metrics were probability of detection, effort, and accuracy. This study showed that the prediction accuracy of this network is higher than logistic regression and discriminant analysis. Also, this network is more cost-effective than ROCKY from the effort perspective. Best metrics set was chosen with logistic regression procedure. They reported that the optimal prediction on KC2 dataset was achieved when the number of metrics was between two and four.

Denaro, Pezzè, and Morasca (2003) used logistic regression with method level metrics on antenna configuration system, Apache 1.3, and Apache 2.0 software to predict the fault-prone modules. Performance evaluation metrics were R^2, completeness, completeness of faulty modules, and correctness of faulty modules. They showed that logistic regression with cross-validation is an effective approach for software fault prediction on industrial projects. Furthermore, they suggested using the cross-validation technique if there is a limited number of data.

Menzies, DiStefano, Orrego, and Chapman (2004) applied Naive Bayes algorithm on public datasets locating in PROMISE repository. Method level metrics were used and probability of detection (PD) and probability of false alarm (PF) were performance evaluation metrics. They stated that accuracy is not an appropriate parameter for evaluation and Naive Bayes provides better performance than J48 algorithm. Furthermore, they reported that PD on KC1 dataset was 55% and PD for Fagan inspections was between 33% and 65%. For industrial inspections, PD for Fagan inspections was between 13% and 30%. Therefore, they suggested using software fault prediction activity in addition to inspection quality assurance activity.

Khoshgoftaar and Seliya (2004) assessed software quality classification techniques such as logistic regression, case based reasoning, classification and regression trees (CART), tree based classification with S-PLUS, Spring-Sliq, C4.5 and Treedisc by using 24 product metrics and four execution metrics on a large telecommunications system. Expected cost of misclassification metric was chosen as performance evaluation parameter. Performance of models changed significantly with respect to different versions of software system. They stated that data and system characteristics affect the performance of prediction models in software engineering.

Wang, Yu, and Zhu (2004) applied artificial neural networks for quality prediction on a large telecommunications system developed with C language by using seven product metrics calculated with MATRIX analyzer. Their objective was to improve the understandability of neural networks based quality prediction models and they applied Clustering Genetic Algorithm (CGA) for this purpose. This algorithm was used to extract rules from artificial neural networks and accuracy parameter was used for performance evaluation. Even though the accuracy of the prediction which was performed with rule set of CGA is lower than the accuracy of the neural networks based prediction, results were more understandable when rule set of CGA was used.

Mahaweerawat, Sophatsathit, Lursinsap, and Musilek (2004) first used multi layer perceptron to identify the fault-prone modules of 3000 C++ classes collected from different web pages and later, they applied radial basis functions (RBF) to classify them with respect to fault types. Metrics were collected with Understand for C++ tool and evaluation metrics were accuracy, Type-I error, Type-II error, inspection, and achieved quality parameters. They decided to collect thousands of classes from numerous developers to build a general prediction model, but prediction models cannot be modeled in such a generic way and that model can highly depend on these 3000 classes. Accuracy, achieved quality, inspection, Type-I error, Type-II error were 90%, 91.53%, 59.55%, 5.32%, and 2.09% respectively. Also, they stated that they could not identify only 2.09% of faulty classes.

Menzies and Di Stefano (2004) investigated linear standard regression (LSR), model trees, ROCKY and Delphi detectors by using method level metrics on public NASA datasets for software fault prediction. Delphi detectors mark modules as fault-prone when one of the following rules is satisfied:

- Cyclomatic complexity is equal to 10 or bigger than 10.
- Design complexity is equal to 4 or bigger than 4.

They suggested using ROCKY with McCabe metrics and they stated that Delphi detectors can be the second choice after ROCKY algorithm. Performance evaluation metrics were accuracy, sensitivity, specificity, precision, and effort.

Kaminsky and Boetticher (2004) predicted faults with genetic algorithm on KC2 dataset by using 21 method level metrics. They investigated the effect of data equalization in this study and *T*-test was used for performance evaluation. They divided data starved domains into two groups: Explicit and Implicit. These domains are explained as follows:

- *Explicit*: Project cost estimation locates in this group. Companies don't want to share their data not to lose their market when they found a good estimation model.
- *Implicit*: Software fault prediction locates in this group. There are adequate data points but datasets are highly imbalanced.

They reported that data equalization is very useful when used together with genetic programming. However, training duration increases because the dataset enlarges after data equalization process is performed. While average fitness value was 12 for normal dataset, it was 81.4 for equalized data and *T*-test showed that this is a very significant difference.

Kanmani, Uthariaraj, Sankaranarayanan, and Thambidurai (2004) applied General Regression Neural Networks (GRNN) technique by using 64 class level metrics on student projects developed in Pondicherry Engineering College for software quality prediction. Principal component analysis was used for feature selection and evaluation parameters were correlation coefficient (r), R^2, average square error, average absolute error, maximum absolute error, and minimum absolute error parameters. They reported that GRNN technique provided good results for fault prediction.

Zhong, Khoshgoftaar, and Seliya (2004) used *K*-means and Neural-Gas clustering methods to cluster modules, and then an expert who is 15 years experienced engineer, labeled each cluster as fault-prone or not fault-prone by examining not only the representative of each cluster, but also some statistical data such as global mean, minimum, maximum, median, 75%, and 90% of each metric. Mean squared error (MSE), average purity, and time parameters were used to evaluate the clustering quality. False positive rate (FPR), false negative rate (FNR), and overall misclassification rate parameters were used to evaluate the performance of expert. According to expert's opinion, it was simpler to label modules by using Neural-Gas clustering. *K*-means clustering worked faster than Neural-Gas and Neural Gas's overall misclassification rate was better than *K*-means clustering's misclassification rate. Neural-Gas performed much better than *K*-means according to the MSE parameter and its average purity was slightly better than *K*-means clustering's purity value.

2.4. Studies between Year 2005 and Year 2007 (17 papers in review)

Xing, Guo, and Lyu (2005) predicted software quality by using Support Vector Machines (SVM) and 11 method level metrics (Halstead, McCabe, Jensen's program length, and Belady's bandwidth) on a medical imagining software. Type-I error and Type-II error were used to evaluate the performance of model. They reported that SVM performed better than quadratic discriminant analysis and classification tree. However, papers published in 2007 showed that SVM does not work well on public datasets and next section will explain them.

Koru and Liu (2005) investigated the effect of module size on fault prediction by using J48 and KStar algorithms. *F*-measure was used for performance evaluation and both method level and class level metrics were investigated on public NASA datasets. They suggested using fault predictors on large components and they advised to use class level metrics instead of method level metrics on small components. A component is identified as large or small with respect to its lines of code. Static measures in small components are near to 0, and therefore, machine learning algorithms cannot work well in such datasets. The best performance was achieved with J48 algorithm and Bayesian Networks, artificial neural networks, and Support Vector Machines did not work well on public datasets. Performances of models were not well when there are very small components in datasets. They stated that machine learning algorithms in WEKA are neither encouraging and nor frustrating.

Khoshgoftaar, Seliya, and Gao (2005) assessed a new three-group software quality classification technique by using C4.5 decision tree, discriminant analysis, case based reasoning, and logistic regression on two embedded software which configures wireless products. They applied five file level metrics and performance evaluation metrics was *expected cost of misclassification*. They reported that three groups such as high, medium, and low labels provided encouraging performance for fault prediction and three-group classification technique let classifiers to classify modules into three classes without changing their designs.

Koru and Liu (2005) built fault prediction models by using J48, K-Star, and Random Forests on public NASA datasets and they used method and class level metrics. *F*-measure was selected as performance evaluation metric. KC1 dataset has method level metrics and they converted them into class level ones by using minimum, maximum, average and sum operations. Therefore, 21 method level metrics were converted into 84 ($21 * 4 = 84$) class level metrics. They stated that large modules had higher *F*-measure values for J48, K-Star and Random Forests algorithms. *F*-measure was 0.65 when they applied class level metrics and when they chose method level metrics, *F*-measure was 0.40. Therefore, they stated that class level metrics improved the model performance, but detection of faults was at class level instead of model level.

Challagulla, Bastani, Yen, and Paul (2005) assessed linear regression, pace regression, support vector regression, neural network for continuous goal field, support vector logistic regression, neural network for discrete goal field, Naive Bayes, instance based learning (IBL), J48, and 1-R techniques by using method level metrics on public NASA datasets for software fault prediction. Performance evaluation metric was average absolute error. They reported that there was no method which provided the best performance in all the datasets. IBR and 1-R was better than the other algorithms according to the accuracy parameter and they stated that principal component analysis did not provide advantage. Furthermore, they explained that size and complexity metrics are not enough for fault prediction new metrics are needed.

Gyimothy, Ferenc, and Siket (2005) studied to validate object oriented metrics for fault prediction by using logistic regression, linear regression, decision trees, and neural networks on Mozilla open source project. Class level metrics were used and performance evaluation metrics were completeness, correctness, and precision. They reported that coupling between object classes (CBO) metric is very useful for fault prediction, four method provide similar results, multivariate models are more useful than lines of code metric, depth of inheritance tree (DIT) metric is not reliable for fault prediction, and number of children (NOC) metric should not be used.

Ostrand, Weyuker, and Bell (2005) predicted the location and number of faults by using negative binomial regression model on two industrial systems and some metrics they used were programming language, age of file, and file change status. Performance evaluation metric was accuracy. They reported that the accuracy of general performance was 84% and the simplified model's accuracy was 73%.

Tomaszewski, Lundberg, and Grahn (2005) studied on the accuracy of early fault prediction in modified code. They applied regression techniques and method/class level metrics on a large telecommunications system and performance evaluation parameter R^2 (determination coefficient). They reported that models built after the system is implemented are 34% more accurate than models built before the system is implemented. Size of changed classes metric can only be calculated after the system implementation and therefore, performance of models improve when this metric is used. When this metric is not preferred, performance values before and after the system implementation are nearly same. Multivariate models were 5% better than univariate regression models and this can be neglected. They suggested using univariate regression models for fault prediction because they are more stable and they do not have multicollinearity feature. This feature exists when there is a correlation between variables and it can be removed with some techniques such as principal component analysis.

Hassan and Holt (2005) tried to identify the top ten fault-prone components on six open source projects. Metrics such as change frequency and size metrics were used. Performance evaluation metrics were hit rate and average prediction age (APA). APA metric was suggested in this study. They proposed some techniques such as most frequently modified (MFM), most recently modified (MRM), most frequently fixed (MFF), and most recently fixed (MRF). MFM and MFF were more successful than the other methods.

Ma, Guo, and Cukic (2006) investigated the performance of numerous machine learning algorithms on public NASA datasets and they used method level metrics. Performance evaluation metrics were G-mean1, G-mean2, and *F*-measure. Area under ROC curve was used to compare the performance of Random Forests with Balanced Random Forests. Logistic regression, discriminant analysis, classification tree, boosting, Kernel Density, Naive Bayes, J48, IBk, Voted Perceptron, VF1, Hyperpipes, ROCKY, Random Forests techniques were investigated in this study. They sorted these algorithms according to their performance results for each performance metric (G-mean1, G-mean2, and *F*-measure) and marked the top three algorithms for each performance metric. They identified the algorithm which provides G-mean1, G-mean2, and *F*-measure in top three. According to this study, Balanced Random Forests is the best algorithm on large datasets such as JM1 and PC1 for software fault prediction problem. They reported that boosting, rule set, single tree classifiers do not provide acceptable results even though these algorithms have been used in literature. Classifiers which have low PD and accuracy were not suggested for fault prediction and balanced random forests' performance was better than the performance of traditional random forests. However, it is hard to find a software tool which has this algorithm. In addition, they suggested using G-mean and *F*-measure parameters to evaluate the performance of imbalanced datasets.

Challagulla, Bastani, and Yen (2006) predicted software faults with Memory Based Reasoning (MBR) on public NASA datasets by using 21 method level metrics. Performance evaluation metrics were probability of detection, probability of false alarm, and accuracy. They proposed a framework and users can choose the MBR configuration which can provide the best performance from this framework.

Khoshgoftaar, Seliya, and Sundaresh (2006) applied case based reasoning by using 24 product and four execution metrics on a large telecommunications system to predict software faults. Performance evaluation metrics were average absolute error and average relative error. They reported that case based reasoning works better than multivariate linear regression and correlation based feature selection and stepwise regression model selection did not improve the performance of models. When CBR was used with Mahalanobis distance, the best performance was achieved. When principal component analysis is applied to remove the metrics' correlations, CBR with city block distance approach provided better results than CBR with Mahalanobis approach.

Nikora and Munson (2006) built high-quality software fault predictors on mission data system of Jet Propulsion Laboratory by using method level metrics. There was no quantitative definition for software faults. For example, some researchers looked at the Discrepancy Reports-DR to identify the faults. However, DR reports show the deviations from software requirements and therefore, they should be called as failures, not faults. In this study, they built a framework which includes the rules for fault definition and they proved that fault predictors which look at the token differences between two versions are more effective. From $a = b + c$ code line, we can extract the following tokens: $B1 = \{<a>, <=>, <b>, <+>, <c>\}$. In next version, if this code line changes to $a = b - c$ code line, the set which shows the differences between two versions will be $B2 = \{<+>, <->\}$. This means that there is one fault between two versions and + was converted to − in new version.

Zhou and Leung (2006) predicted high and low severity faults by using object oriented software metrics on NASA's KC1 dataset. They investigated logistic regression, Naive Bayes, Random Forests, nearest neighbor with generalization techniques for fault prediction. Performance evaluation metrics were correctness, completeness, and precision. They reported that low severity faults can predicted with a better performance compared to high severity faults and number of children (NOC) is not significant for high severity faults prediction. They stated the other Chidamber–Kemerer metrics were useful for fault prediction.

Mertik, Lenic, Stiglic, and Kokol (2006) estimated software quality with prunned C4.5, unprunned C4.5, multimethod, SVM using RBF kernel, SVM using linear kernel techniques by using method level metrics on NASA datasets. Multimethod data mining tool was developed in Maribor University and it shows the results in tree format. Multimethod includes several methods and its performance with respect to performance was very high.

Boetticher (2006) investigated the effects of datasets on software engineering and applied J48 and Naive Bayes techniques for the analysis on NASA public datasets. Performance evaluation metrics were PD, PF, and accuracy. Datasets were divided into three parts: training set, nice neighbors test set, and nasty neighbors test set. Nice neighbors are neighbors who are close to the same class and nasty neighbors are neighbors who locate in different classes. He showed that the accuracy was 94% for nice neighbors test set and the accuracy was 20% for nasty neighbors test set. Also, he stated that ten-fold cross-validation is not enough for validation and the difficulty of datasets is a critical issue for evaluation.

Bibi, Tsoumakas, Stamelos, and Vlahvas (2008) used Regression via Classification (RvC) technique on Pekka dataset which was collected in one of Finland's banks and they used different metrics such as disc usage, processor usage, number of users, and document quality for software fault prediction. Performance evaluation metrics were average absolute error and accuracy. They reported that RvC can be used to enhance the understandability of regression models.

2.5. Studies between Year 2007 and Year 2009 (33 papers in review)

Gao and Khoshgoftaar (2007) investigated the performance of Poisson regression, zero-inflated poisson regression, negative binomial regression model, Zero-Inflated negative binomial, and Hurdle regression (HP1, HP2, HNB1, HNB2) techniques on two embedded software applications which configure the wireless telecommunications products by using five file level metrics for software fault prediction. Performance evaluation metrics were Pearson's chi square measure, information criteria, average absolute error (AAE), and average relative error (ARE). They reported that model based on Zero-Inflated negative binomial technique performs better than the other algorithms according to the information criteria and chi square measures. Model based on HP2 technique was the best one when AAE and ARE parameters were used.

Li and Reformat (2007) predicted software faults with SimBoost technique on JM1 dataset by using method level metrics and accuracy parameter was used for performance evaluation. SimBoost method was proposed in this study and fuzzy labels for classification were suggested. SimBoost did not work well at first and finally fuzzy labels were used. This method with fuzzy labels worked well on this dataset.

~~Mahaweerawat, Sophatsathit, and Lursinsap (2007)~~ first used self organizing map clustering and then applied RBF for software fault prediction on a dataset which is not explained in the paper. Method level metrics were used and performance evaluation metric was mean of absolute residual (MAR). MAR is calculated by subtracting predicted number of faults from the actual number of faults and taking the absolute value of this result. They reported that accuracy was 93% in this study but accuracy is not an appropriate parameter for imbalanced datasets.

Menzies et al. (2007a) investigated several data mining algorithms for software fault prediction on public NASA datasets and they used method level metrics. Performance evaluation metrics were PD, PF, and balance. They achieved the best performance Naive Bayes algorithm and before this algorithm is applied, they used logNum filter for software metrics. They reported that Naive Bayes outperformed J48, the best algorithm changed according to the dataset characteristics, one dataset and one learning algorithm is not enough for software fault prediction and numerous experiments should be performed for a robust prediction model.

Zhang and Zhang (2007) criticized Menzies et al.'s study (2007a) and they stated that PD and PF parameters are not enough to evaluate the performance of models. They reported that precision was very low in Menzies et al.'s study (2007a) and this model would not be very useful in practice.

Menzies, Dekhtyar, Distefano, and Greenwald (2007b) responded to comments of Zhang and Zhang (2007) and they stated that precision is not a useful parameter for software engineering problems, models which have low precision provided remarkable results for different problems. Precision is supposed to be high too, but in practice this is not a real case. Furthermore, they reported that some papers which provide models with low precision were selected as best paper in conferences and these models are very useful even though they have low precision.

Ostrand, Weyuker, and Bell (2007) predicted fault-prone modules with negative binomial regression model by using several metrics such as file size, file status, the number of changes on file, and programming language. They reported that negative binomial regression model is very useful according to the accuracy parameter.

Yang, Yao and Huang (2007) applied Fuzzy Self-Adaptation Learning Control Network-(FALCON) on an artificial dataset. FALCON is a special kind of fuzzy neural network. Inputs for FALCON were software metrics such as complexity, reuse, and depth of inheritance tree and outputs were reliability and effectiveness. They stated that this model can measure several quality features such as reliability, performance, and maintainability, but they did not try this approach with real datasets. Therefore, it is clear that their study is at early stages.

Pai and Dugan (2007) used linear regression, Poisson regression, and logic regression to calculate conditional probability densities of Bayes Networks' nodes and then, they used these networks to calculate the fault proneness of modules on NASA datasets. Chidamber–Kemerer metrics and lines of code metric were used and evaluation parameters were sensitivity, specificity, precision, false positive and false negative parameters. They reported that weighted methods count (WMC), coupling between object classes (CBO), response for a class (RFC), and lines of code metrics are very useful to predict the fault-proneness of modules. Poisson regression analysis showed that depth of inheritance tree (DIT) and numbers of children (NOC) are not significant, but lacks of cohesion of methods (LCOM) are significant for fault-proneness. Linear model which does not include LCOM metric was better than the Poisson model which includes LCOM metric. Also, they stated that Bayes model can combine product and process metrics and a model which uses both of these metrics were suggested.

Wang, Zhu, and Yu (2007) predicted software quality with genetic algorithm by using 18 method level and 14 file level metrics on two large telecommunications system. Performance evaluation metrics were Type-I error, Type-II error, and overall misclassification rate. They reported that the proposed model is better than S-PLUS and TreeDisc. They removed the clusters which have less than five members and when purity was more than 80%, clusters were labeled with the dominant class label.

Seliya and Khoshgoftaar (2007a) predicted software faults with limited fault data by using Expectation–Maximization (EM) technique. JM1 dataset was used as training dataset and KC1, KC2, and KC3 were used as test datasets. The importance of this study was to propose a model for limited fault data problem. Sometimes we may not have enough fault data to build accurate models. For example, some project partners may not collect fault data for some project components or execution cost of metrics collection tools on the whole system may be extremely expensive. In these situations, we need powerful classifiers which can build accurate classification models with limited fault data or powerful semi-supervised classification algorithms which can benefit from both unlabeled data and labeled data. They used Type-I error, Type-II error, and overall misclassification rate parameters for performance evaluation. EM technique is first used to give labels to unlabeled data and then all the modules are used to build the prediction model. EM based quality model provided acceptable results for semi-supervised fault prediction problem.

Koru and Liu (2007) identified change-prone classes with tree based models on K-Office and Mozilla open source projects by using class label metrics. 20% of classes changed and tree based models were very useful to identify these change-prone classes.

~~Cukic and Ma (2007)~~ predicted fault-proneness of modules and investigated 16 learning algorithms on JM1 dataset by using method level metrics. Evaluation parameters were PD and PF. Only four algorithms provided PD value which is higher than 50% and PF value which is lower than 50%. They reported that it is highly critical to use public datasets and common performance evaluation parameters for software fault prediction studies.

Tomaszewski, Håkansson, Grahn, and Lundberg (2007) used expert opinion and univariate linear regression analysis to predict software faults by using method and class level metrics on two software systems developed in Ericsson. Accuracy parameter was used for performance evaluation. Eleven experts were invited to this study and they predicted fault-proneness of components. They reported that statistical approaches are more successful than expert opinion and experts did not predict faults easily in large datasets. The accuracy of prediction model based on class level metrics was better than the model based on component level metrics. While statistical approaches are cheaper than expert based approach, expert opinions can consider the project specific issues during prediction. They stated that different experts choose different components as fault-prone, and even if they choose the same components as fault-prone, they calculate diverse fault density values for components. Also, expert experience on products did not affect the results.

Seliya and Khoshgoftaar (2007b) proposed a constraint-based semi-supervised clustering scheme that uses *K*-means clustering method to predict the fault-proneness of program modules when the defect labels for modules are unavailable. However, their approach uses an expert's domain knowledge to iteratively label clusters as fault-prone or not. The expert labels clusters mainly basing his predictions on his knowledge and some statistical information. This means that this study does not present a well-defined decision mechanism of the expert. On the other hand, the enlargement of the dataset causes the increasing of the number of clusters and iterations, which will require to the expert to spend much more time on this approach. The expert necessity prevents this method to be automatically performed, presenting one of the major drawbacks of this study. Performance evaluation metrics were Type-I error, Type-II error, and overall misclassification rate. They reported that semi-supervised clustering schema provided better performance than traditional clustering methods and half of the modules which could not be labeled were noisy.

Olague, Gholston, and Quattlebaum (2007) investigated three software metrics suites on open source Rhino project's six versions to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes. Univariate binary logistic regression (UBLR) and multivariate binary logistic regression (MBLR) techniques were used for the analysis. Chidamber–Kemerer (CK) metrics suite, Abreu's object oriented metrics (MOOD), and Bansiya and Davis's quality metrics (QMOOD) were investigated. Accuracy parameter was used for model validation and Spearman correlation was applied to examine the metrics' effects. They reported that CK and QMOOD metrics are very useful for fault prediction, but MOOD metrics are useless. Furthermore, they stated that MBLR models are useful for iterative and agile software development processes, WMC (weighted methods count) and RFC (response for children) metrics from CK metrics suite are useful for fault prediction, CIS and NOM metrics from QMOOD metrics suite are useful according to the UBLR analysis. The best performance was achieved when CK metrics were used.

Binkley, Feild, and Lawrie (2007) used linear mixed-effects regression model on Mozilla and a commercial product by using QALP score, total lines of code, and lines of code except comments and blank lines. Determination coefficient was used as performance evaluation parameter. They reported that neither QALP nor size measure is good predictor for Mozilla project.

Jiang, Cukic, and Menzies (2007) predicted software faults using early lifecycle data. They investigated 1-R, Naive Bayes, Voted Perceptron, Logistic Regression, J48, VFI, IBk, and Random Forests for fault prediction by using metrics extracted from textual requirements and code metrics. PD and PF were selected as performance evaluation metrics. They reported that the performances of algorithms except Voted Perceptron algorithm improved when code metrics were combined with requirement metrics. Only three projects of NASA include requirement metrics and ARM (Automated Requirement Measurement) tool was developed in NASA Goddard Software Assurance Research Center to extract requirement metrics from requirement documents.

Bibi et al. (2008) applied Regression via Classification (RvC) to software fault prediction problem to estimate the number of software faults with a confidence interval. RvC provided better regression error than the standard regression methods. Pekka dataset from a big commercial bank in Finland and ISBSG (International Software Benchmarking Standards Group) dataset were used for the analysis. Performance evaluation metric was Mean Absolute Error (MAE).

Bingbing, Qian, Shengyong, and Ping (2008) used Affinity Propagation clustering algorithm on two datasets and compared the performance of it with the performance of *K*-means clustering method. Affinity Propagation was developed by Brendan J. Frey and Delbert Dueck. This algorithm was better than *K*-means clustering on two datasets according to the Type-II error. Performance evaluation metrics were Type-I error, Type-II error, and entire correct classification rate (CCR). A medical imaging system and Celestial Spectrum Analysis System datasets were used for the analysis. Affinity Propagation reduced Type-II error and increased CCR on software datasets and the number of cluster was two.

Marcus, Poshyvanyk, and Ferenc (2008) proposed a new cohesion metric named Conceptual Cohesion of Classes (C3) in this study. They showed that the combination of structural and conceptual cohesion metrics provide better performance than the standalone usage of structural metrics for software fault prediction modeling. WinMerge and Mozilla projects were used for experimental evaluation. Performance evaluation metrics were precision, correctness, and completeness. Univariate logistic regression analysis showed that C3 ranks better than many of the cohesion metrics.

Shafi, Hassan, Arshaq, Khan, and Shamail (2008) compared the performance of 30 techniques on two datasets for software quality prediction. First dataset is JEditData and second one is AR3 from PROMISE repository. Performance evaluation parameters were precision, recall, specificity, and accuracy. This study showed that the techniques classification via regression and LWL performed better than the other techniques. However, only two datasets were used in this study.

Riquelme, Ruiz, Rodríguez, and Moreno (2008) investigated two balancing techniques with two classifiers, Naive Bayes and C4.5, on five public datasets from PROMISE repository for software fault prediction. They reported that balancing techniques improve the AUC measure, but did not improve the percentage of correctly classified instances. Performance evaluation metrics were AUC and percentage of correctly classified instances. Sampling metrics were *resample* implementation of WEKA and SMOTE.

Shatnawi and Li (2008) studied the effectiveness of software metrics in identifying fault-prone classes in post-release software evolution process. Three releases of Eclipse project were investigated and the prediction accuracy decreased from release to release. Performance evaluation metric was AUC. They predicted class error-proneness and error severity categories in the post-release evolution of a system. Error severity categories were more useful than two class categorization because more severe faults were identified with severity categories.

Arisholma, Briand, and Johannessen (2010) evaluated fault-proneness models on a large Java legacy system project. They reported that modeling technique has limited affect on the prediction accuracy, process metrics are very useful for fault prediction, and the best model is highly dependent on the performance evaluation parameter. They proposed a surrogate measure of cost-effectiveness for assessment of models. Adaboost combined with C4.5 provided the best results and techniques were used with default parameters.

Catal and Diri (2009a) focused on the high-performance fault predictors based on machine learning such as Random Forests and algorithms based on a new computational intelligence approach called Artificial Immune Systems. Public NASA datasets were used. They reported that Random Forests provides the best prediction performance for large datasets and Naive Bayes is the best prediction algorithm for small datasets in terms of the Area under Receiver Operating Characteristics Curve (AUC) evaluation parameter.

Turhan and Bener (2009) showed that independence assumption in Naive Bayes algorithm is not detrimental with principal component analysis (PCA) pre-processing and they used PD, PF and balance parameters in their study.

Chang, Chu, and Yeh (2009) proposed a fault prediction approach based on association rules to discover fault patterns. They reported that prediction results were excellent. The benefit of this method is the discovered fault patterns can be used in causal analysis to find out the causes of faults.

Mende and Koschke (2009) evaluated lines of code metric based prediction on thirteen NASA datasets and this model performed well in terms of Area under ROC curve (AUC) parameter and they could not show statistical significant differences to some data mining algorithms. When effort-sensitive performance measure was used, a line of code metric based prediction approach was the worst one. Thirteen datasets from NASA were used in this study.

Tosun, Turhan, and Bener (2009) conducted experiments on public datasets to validate Zimmermann and Nagappan's paper published in ICSE'08. Three embedded software projects were used for the analysis. They reported that network measures are significant indicators of fault-prone modules for large systems. Performance evaluation metrics were PD, PF, and precision.

Turhan, Kocak, and Bener (2009) investigated 25 projects of a telecommunication system and trained models on NASA MDP data. They used static call graph based ranking (CBGR) and nearest neighbor sampling to build defect predictors. They reported that at least 70% of faults can be identified by inspecting only 6% of code with Naive Bayes model and 3% of code with CBGR model. Twenty-nine static code metrics were used in this research. Cross-company NASA data were used due to the lack of local fault data.

We published three papers on practical software fault prediction problems in 2009 and they are explained in Chapter 3 (Alan & Catal, 2009; Catal & Diri, 2009b; Catal, Sevim, & Diri, 2009).

3. Current trends

3.1. Practical software fault prediction problems

Most of the studies in literature assume that there is enough fault data to build the prediction models. However, there are cases when previous fault data are not available, such as when a company deals with a new project type or when fault labels in previous releases may have not been collected. In these cases, we need new models to predict software faults. This research problem can be called as *software fault prediction of unlabeled program modules*. In a research project started in April 2008, we developed a technique to solve this problem (Catal et al., 2009). Our technique first applies *X*-means clustering method to cluster modules and identifies the best cluster number. After this step, the mean vector of each cluster is checked against the metrics thresholds vector. A cluster is predicted as fault-prone if at least one metric of the mean vector is higher than the threshold value of that metric. In addition to *X*-means clustering-based method, we made experiments with pure metrics thresholds method, fuzzy clustering, and *K*-means clustering-based methods. Experiments reveal that unsupervised software fault prediction can be fully automated and effective results can be produced using *X*-means clustering with software metrics thresholds. Three datasets, collected from Turkish white-goods manufacturer developing embedded controller software, have been used for the validation.

Supervised classification algorithms in machine learning can be used to build the prediction model with previous software metrics and previous fault labels. However, sometimes we cannot have enough fault data to build accurate models. For example, some project partners may not collect fault data for some project components or execution cost of metrics collection tools on the whole system may be extremely expensive. In these situations, we need powerful classifiers which can build accurate classification models with limited fault data or powerful semi-supervised classification algorithms which can benefit from unlabeled data together with labeled one. This research problem can be called as software fault prediction with limited fault data. In the research project started in April 2008, we showed that Naive Bayes algorithm is the best choice to build a semi-supervised fault prediction model for small datasets and YATSI (yet another two stage idea) algorithm may improve the performance of Naive Bayes for large datasets (Catal & Diri, 2009b).

These two research problems are investigated in our research project started in 2008 and this research project will finish on April 2010. We developed an Eclipse based plug-in for software fault prediction and these two problems.

Third research problem for our research project is outlier detection in software measurement datasets. We developed a simple but accurate technique for outlier detection. For this algorithm, the data object which has not-faulty class label is an outlier if the majority of metrics (three or more of six metrics) exceeds their corresponding thresholds values. Also, data object which has faulty class label is an outlier if all the metrics are below of their corresponding thresholds levels (Alan & Catal, 2009).

We need to conduct more research on these three research problems in software engineering community to build more robust and practical prediction models. On the other hand, supervised software fault prediction research problem was extensively studied in literature and Menzies et al. (2007a) showed that Naive Bayes provides the best performance for fault prediction. In addition to software fault prediction problem, researchers should conduct experiments on different prediction problems such as reusability prediction and security prediction.

3.2. Significant researchers

It is very important to know the researchers who contributed so much and who are very active on a research area and in this section; we explain these researchers for software fault prediction area according to our literature review. Tim Menzies, Bojan Cukic, Gunes Koru, Thomas Ostrand, Taghi Khoshgoftaar, Naeem Seliya, Marek Reformat, Burak Turhan, Ayse Bener, John Munson, and Ping Guo are active researchers on software fault prediction. Tim Menzies and Gunes Koru always use public NASA datasets from PROMISE repository.

4. Conclusion

This paper surveyed the software engineering literature on software fault prediction and both machine learning based and statistical based approaches were included in this survey. According to this survey, most of the studies used method level metrics and models were mostly based machine learning techniques. We suggest conducting more research on practical software fault prediction problems explained in Chapter 3 because there are just a few papers about these problems yet. Naive Bayes is a robust machine learning algorithm for supervised software fault prediction problem. Furthermore, we need software tools to automate this prediction process. After development of these tools, fault prediction models will achieve widespread applicability in the software industry. Papers explained in this article reflect the outline of what was published so far, but naturally this is not a complete review of all the papers published so far. This paper will help researchers to investigate the previous studies from metrics, methods, datasets, performance evaluation metrics, and experimental results perspectives in an easy and effective manner.

Acknowledgements

This study is supported by The Scientific and Technological Research Council of TURKEY (TUBITAK) under Grant 107E213. The findings and opinions in this study belong solely to the authors, and are not necessarily those of the sponsor.

References

Alan, O., & Catal, C. (2009). An outlier detection algorithm based on object-oriented metrics thresholds. In *Proceedings of ISCIS 2009 conference, Guzelyurt, 14–16 September 2009, Northern Cyprus*.

Arisholma, E., Briand, L. C., & Johannessen, E. B. (2010). A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *Journal of Systems and Software, 83*(1), 2–17.

Bibi, S., Tsoumakas, G., Stamelos, I., & Vlahvas, I. (2008). Regression via classification applied on software defect estimation. *Expert Systems with Applications, 34*(3), 2091–2101.

Bingbing, Y., Qian, Y., Shengyong, X., & Ping, G. (2008). Software quality prediction using affinity propagation algorithm. In *IJCNN 2008* (pp. 1891–1896).

Binkley, A. B., & Schach, S. R. (1998). Prediction of run-time failures using static product quality metrics. *Software Quality Journal, 7*(2), 141–147.

Binkley, D., Feild, H., & Lawrie, D. (2007). *Software fault prediction using language processing. Testing: Industrial and academic conference. Practice and research techniques*. Cumberland Lodge, Windsor, UK: IEEE Press.

Boetticher, G. (2006). Improving credibility of machine learner models in software engineering. *Advanced machine learner applications in software engineering. Series on software engineering and knowledge engineering*. Hershey, PA, USA: Idea Group Publishing.

Briand, L., Basili, V., & Hetmanski, C. (1993). Developing interpretable models with optimized set reduction for identifying high risk software components. *IEEE Transactions on Software Engineering, 19*(11), 1028–1044.

Catal, C., & Diri, B. (2009a). Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. *Information Sciences, 179*(8), 1040–1058.

Catal, C., & Diri, B. (2009b). Unlabelled extra data do not always mean extra performance for semi-supervised fault prediction. *Expert Systems, 26*, 458–471.

Catal, C., Sevim, U., & Diri, B. (2009). Software fault prediction of unlabeled program modules. In *Proceedings of the world congress on engineering (WCE 2009), 1–3 July 2009, London, UK* (Vol. I).

Challagulla, V. U. B., Bastani, F. B., & Yen, I. L. (2006). A unified framework for defect data analysis using the MBR technique. In *Eighteenth IEEE international conference on tools with artificial intelligence, Washington, DC, USA* (pp. 39–46).

Challagulla, V. U., Bastani, F. B., Yen, I., & Paul, R. A. (2005). Empirical assessment of machine learning based software defect prediction techniques. In *Tenth IEEE international workshop on object-oriented real-time dependable systems* (pp. 263–270). Sedona, Arizona: IEEE Computer Society.

Chang, C., Chu, C., & Yeh, Y. (2009). Integrating in-process software defect prediction with association mining to discover defect pattern. *Information and Software Technology, 51*(2), 375–384.

Cohen, W. W., & Devanbu, P. T. (1997). A comparative study of inductive logic programming methods for software fault prediction. In *Fourteenth international conference on machine learning, Nashville, Tennessee, USA* (pp. 66–74).

Cukic, B., & Ma, Y., (2007). Predicting fault-proneness: Do we finally know how? In *Reliability analysis of system failure data, Cambridge, UK*.

De Almeida, M. A., & Matwin, S. (1999). Machine learning method for software quality model building. In *Eleventh international symposium on foundations of intelligent systems, Warsaw, Poland* (pp. 565–573).

Denaro, G. (2000). Estimating software fault-proneness for tuning testing activities. In *Twenty-second international conference on software engineering* (pp. 704–706). New York, NY: ACM.

Denaro, G., Lavazza, L., & Pezzè, M. (2003). An empirical evaluation of object oriented metrics in industrial setting. In *The 5th CaberNet plenary workshop, Porto Santo, Madeira Archipelago, Portugal*.

Denaro, G., Pezzè, M., & Morasca, S. (2003). Towards industrially relevant fault-proneness models. *International Journal of Software Engineering and Knowledge Engineering, 13*(4), 395–417.

Emam, K. E., Melo, W., & Machado, J. C. (2001). The prediction of faulty classes using object-oriented design metrics. *Journal of Systems and Software, 56*(1), 63–75.

Evett, M., Khoshgoftar, T., Chien, P., & Allen, E. (1998).GP-based software quality prediction. In *Third annual conference on genetic programming* (pp. 60–65).

Gao, K., & Khoshgoftaar, T. M. (2007). A comprehensive empirical study of count models for software fault prediction. *IEEE Transactions for Reliability, 56*(2), 223–236.

Guo, L., Cukic, B., & Singh, H. (2003). Predicting fault prone modules by the Dempster–Shafer belief networks. In *Eighteenth IEEE international conference on automated software engineering* (pp. 249–252). Montreal, Canada: IEEE Computer Society.

Guo, P., & Lyu, M. R. (2000). Software quality prediction using mixture models with EM algorithm. In *First Asia-Pacific conference on quality software* (pp. 69–80). Hong Kong, China: IEEE Computer Society.

Gyimothy, T., Ferenc, R., & Siket, I. (2005). Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software Engineering, 31*(10), 897–910.

Hassan, A. E., & Holt, R. C. (2005). The top ten list: Dynamic fault prediction. In *Twenty-first IEEE international conference on software maintenance* (pp. 263–272). Budapest, Hungary: IEEE Computer Society.

Jiang, Y., Cukic, B., & Menzies, T. (2007). Fault prediction using early lifecycle data. In *Eighteenth IEEE international symposium on software reliability* (pp. 237–246). Trollhättan, Sweden: IEEE Computer Society.

Kaminsky, K., & Boetticher, G. (2004). How to predict more with less, defect prediction using machine learners in an implicitly data starved domain. In *The 8th world multiconference on systemics, cybernetics and informatics, Orlando, FL*.

Kanmani, S., Uthariaraj, V. R., Sankaranarayanan, V., & Thambidurai, P. (2004). Object oriented software quality prediction using general regression neural networks. *SIGSOFT Software Engineering Notes, 29*(5), 1–6.

Kaszycki, G. (1999). Using process metrics to enhance software fault prediction models. In *Tenth international symposium on software reliability engineering, Boca Raton, Florida*.

Khoshgoftaar, T. M. (2002). Improving usefulness of software quality classification models based on Boolean discriminant functions. In *Thirteenth international symposium on software reliability engineering* (pp. 221–230). Annapolis, MD, USA: IEEE Computer Society.

Khoshgoftaar, T. M., & Seliya, N. (2002a). Software quality classification modeling using the SPRINT decision tree algorithm. In *Fourth IEEE international conference on tools with artificial intelligence* (pp. 365–374). Washington, DC: IEEE Computer Society.

Khoshgoftaar, T. M., & Seliya, N. (2002b). Tree-based software quality estimation models for fault prediction. In *Eighth IEEE symposium on software metrics, Ottawa, Canada* (pp. 203–215).

Khoshgoftaar, T. M., & Seliya, N. (2003). Fault prediction modeling for software quality estimation: Comparing commonly used techniques. *Empirical Software Engineering, 8*(3), 255–283.

Khoshgoftaar, T. M., & Seliya, N. (2004). Comparative assessment of software quality classification techniques: An empirical case study. *Empirical Software Engineering, 9*(3), 229–257.

Khoshgoftaar, T. M., Allen, E. B., & Busboom, J. C. (2000). Modeling software quality: The software measurement analysis and reliability toolkit. In *Twelfth IEEE international conference on tools with artificial intelligence* (pp. 54–61). Vancouver, BC, Canada: IEEE Computer Society.

Khoshgoftaar, T. M., Allen, E. B., Hudepohl, J. P., & Aud, S. J. (1997). Application of neural networks to software quality modeling of a very large telecommunications system. *IEEE Transactions on Neural Networks, 8*(4), 902–909.

Khoshgoftaar, T. M., Geleyn, E., & Gao, K. (2002). An empirical study of the impact of count models predictions on module-order models. In *Eighth international symposium on software metrics* (pp. 161–172). Ottawa, Canada: IEEE Computer Society.

Khoshgoftaar, T. M., Seliya, N., & Gao, K. (2005). Assessment of a new three-group software quality classification technique: An empirical case study. *Empirical Software Engineering, 10*(2), 183–218.

Khoshgoftaar, T. M., Seliya, N., & Sundaresh, N. (2006). An empirical study of predicting software faults with case based reasoning. *Software Quality Journal, 14*(2), 85–111.

Khoshgoftaar, T., Gao, K., & Szabo, R. M. (2001). An application of zero-inflated poisson regression for software fault prediction. In *Twelfth international symposium on software reliability engineering* (pp. 66–73). Washington, DC: IEEE Computer Society.

Koru, A. G., & Liu, H. (2005a). An investigation of the effect of module size on defect prediction using static measures. In *Workshop on predictor models in software engineering* (pp. 1–5). Missouri: St. Louis.

Koru, A. G., & Liu, H. (2007). Identifying and characterizing change-prone classes in two large-scale open-source products. *Journal of Systems and Software, 80*(1), 63–73.

Koru, A. G., & Tian, J. (2003). An empirical comparison and characterization of high defect and high complexity modules. *Journal of Systems and Software, 67*(3), 153–163.

Koru, G., & Liu, H. (2005b). Building effective defect prediction models in practice. *IEEE Software, 22*(6), 23–29.

Lanubile, F., Lonigro, A., & Visaggio, G. (1995). Comparing models for identifying fault-prone software components. In *Seventh international conference on software engineering and knowledge engineering* (pp. 312–319).

Li, Z., & Reformat M. (2007). A practical method for the software fault-prediction. In *IEEE international conference on information reuse and integration, Las Vegas, Nevada, USA* (pp. 659–666).

Ma, Y., Guo, L., & Cukic, B. (2006). *A statistical framework for the prediction of fault-proneness. Advances in machine learning application in software engineering*. Idea Group Inc.. pp. 237–265.

Mahaweerawat, A., Sophasathit, P., & Lursinsap, C. (2002). Software fault prediction using fuzzy clustering and radial basis function network. In *International conference on intelligent technologies, Vietnam* (pp. 304–313).

Mahaweerawat, A., Sophatsathit, P., & Lursinsap, C. (2007). Adaptive self-organizing map clustering for software fault prediction. In *Fourth international joint conference on computer science and software engineering, Khon Kaen, Thailand* (pp. 35–41).

Mahaweerawat, A., Sophatsathit, P., Lursinsap, C., & Musilek, P. (2004). Fault prediction in object-oriented software using neural network techniques. In *Proceedings of the InTech conference, Houston, TX, USA* (pp. 27–34).

Marcus, A., Poshyvanyk, D., & Ferenc, R. (2008). Using the conceptual cohesion of classes for fault prediction in object-oriented systems. *IEEE Transactions Software Engineering, 34*(2), 287–300.

Mende, T., & Koschke, R. (2009). Revisiting the evaluation of defect prediction models. In *Proceedings of the 5th international conference on predictor models in software engineering (Vancouver, British Columbia, Canada, May 18–19, 2009). PROMISE '09* (pp. 1–10). New York, NY: ACM.

Menzies, T., & Di Stefano, J. S. (2004). How good is your blind spot sampling policy? In *Eighth IEEE international symposium on high-assurance systems engineering* (pp. 129–138). Tampa, FL, USA: IEEE Computer Society.

Menzies, T., Dekhtyar, A., Distefano, J., & Greenwald, J. (2007b). Problems with precision: A response to comments on data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering, 33*(9), 637–640.

Menzies, T., DiStefano, J., Orrego, A., & Chapman, R. (2004). Assessing predictors of software defects. In *Predictive software models workshop*.

Menzies, T., Greenwald, J., & Frank, A. (2007a). Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering, 33*(1), 2–13.

Mertik, M., Lenic, M., Stiglic, G., & Kokol, P. (2006). Estimating software quality with advanced data mining techniques. In *International conference on software engineering advances* (pp. 19). Papeete, Tahiti, French Polynesia: IEEE Computer Society.

Nikora, A. P., & Munson, J. C. (2006). Building high-quality software fault predictors. *Software-Practice and Experience, 36*(9), 949–969.

Ohlsson, N., Zhao, M., & Helander, M. (1998). Application of multivariate analysis for software fault prediction. *Software Quality Journal, 7*(1), 51–66.

Olague, H. M., Gholston, S., & Quattlebaum, S. (2007). Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes. *IEEE Transactions on Software Engineering, 33*(6), 402–419.

Ostrand, T. J., Weyuker, E. J., & Bell, R. M. (2005). Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering, 31*(4), 340–355.

Ostrand, T. J., Weyuker, E. J., & Bell, R. M. (2007). Automating algorithms for the identification of fault-prone files. In *International symposium on software testing and analysis, London, United Kingdom* (pp. 219–227).

Pai, G. J., & Dugan, J. B. (2007). Empirical analysis of software fault content and fault proneness using Bayesian methods. *IEEE Transactions on Software Engineering, 33*(10), 675–686.

Pizzi, N. J., Summers, R., & Pedrycz, W. (2002). Software quality prediction using median-adjusted class labels. In *International joint conference on neural networks* (pp. 2405–2409). Honolulu, HI: IEEE Computer Society.

Porter, A. A., & Selby, R. W. (1990). Empirically guided software development using metric-based classification trees. *IEEE Software, 7*(2), 46–54.

Reformat, M. (2003). A fuzzy-based meta-model for reasoning about number of software defects. In *Tenth international fuzzy systems association world congress, Istanbul, Turkey* (pp.644–651).

Riquelme, J. C., Ruiz, R., Rodríguez, D., & Moreno, J. (2008). Finding defective modules from highly unbalanced datasets. *Actas del 8° taller sobre el apoyo a la decisión en ingeniería del software* (ADIS-JISBD) (pp. 67–74).

Schneidewind, N. F. (2001). Investigation of logistic regression as a discriminant of software quality. In *Seventh international symposium on software metrics* (pp. 328–337). Washington, DC: IEEE Computer Society.

Seliya, N., & Khoshgoftaar, T. M. (2007a). Software quality estimation with limited fault data: A semi supervised learning perspective. *Software Quality Journal, 15*(3), 327–344.

Seliya, N., & Khoshgoftaar, T. M. (2007b). Software quality analysis of unlabeled program modules with semisupervised clustering. *IEEE Transactions on Systems, Man, and Cybernetics, 37*(2), 201–211.

Shafi, S., Hassan, S. M., Arshaq, A., Khan, M. J., & Shamail, S. (2008). Software quality prediction techniques: A comparative analysis. In *Fourth international conference on emerging technologies* (pp. 242–246).

Shatnawi, R., & Li, W. (2008). The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process. *Journal of Systems and Software, 81*(11), 1868–1882.

Thwin, M. M., & Quah, T. (2003). Application of neural networks for software quality prediction using object-oriented metrics. In *Nineteenth international conference on software maintenance* (pp. 113–122). Amsterdam, The Netherlands: IEEE Computer Society.

Tomaszewski, P., Håkansson, J., Grahn, H., & Lundberg, L. (2007). Statistical models vs. expert estimation for fault prediction in modified code – An industrial case study. *Journal of Systems and Software, 80*(8), 1227–1238.

Tomaszewski, P., Lundberg, L., & Grahn, H. (2005). The accuracy of early fault prediction in modified code. In *Fifth conference on software engineering research and practice in Sweden, Västerås, Sweden* (pp. 57–63).

Tosun, A., Turhan, B., & Bener, A. (2009). Validation of network measures as indicators of defective modules in software systems. In *Proceedings of the 5th international conference on predictor models in software engineering* (Vancouver, British Columbia, Canada, May 18–19, 2009). *PROMISE '09* (pp. 1–9). New York, NY: ACM.

Turhan, B., & Bener, A. (2009). Analysis of Naive Bayes' assumptions on software fault data: An empirical study. *Data Knowledge Engineering, 68*(2), 278–290.

Turhan, B., Kocak, G., & Bener, A. (2009). Data mining source code for locating software bugs: A case study in telecommunication industry. *Expert Systems and Application, 36*(6), 9986–9990.

Wang, Q., Yu, B., & Zhu, J. (2004). Extract rules from software quality prediction model based on neural network. In *Sixteenth IEEE international conference on tools with artificial intelligence* (pp. 191–195). Boca Raton, FL, USA: IEEE Computer Society.

Wang, Q., Zhu, J., & Yu, B. (2007). Feature selection and clustering in software quality prediction. In *Eleventh international conference on evaluation and assessment in software engineering, Keele, England*.

Xing, F., Guo, P., & Lyu, M. R. (2005). A novel method for early software quality prediction based on support vector machine. In *Sixteenth IEEE international symposium on software reliability engineering* (pp. 213–222). Chicago, IL, USA: IEEE Computer Society.

Xu, Z., Khoshgoftaar, T. M., & Allen, E. B. (2000). Prediction of software faults using fuzzy nonlinear regression modeling. In *Fifth IEEE international symposium on high assurance systems engineering, Albuquerque, New Mexico* (pp. 281–290).

Yang, B., Yao, L., & Huang, H. Z. (2007). Early software quality prediction based on a fuzzy neural network model. In *Third international conference on natural computation, Haikou, China* (pp. 760–764).

Yuan, X., Khoshgoftaar, T. M., Allen, E. B., & Ganesan, K. (2000). An application of fuzzy clustering to software quality prediction. In *Third IEEE symposium on application-specific systems and software engineering technology* (pp. 85). Washington, DC: IEEE Computer Society.

Zhang, H., & Zhang, X. (2007). Comments on data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering, 33*(9), 635–637.

Zhong, S., Khoshgoftaar, T. M., & Seliya, N. (2004). Unsupervised learning for expert-based software quality estimation. In *Eighth IEEE international symposium on high assurance systems engineering* (pp. 149–155).

Zhou, Y., & Leung, H. (2006). Empirical analysis of object-oriented design metrics for predicting high and low severity faults. *IEEE Transactions on Software Engineering, 32*(10), 771–789.