Software Defect Prediction: Review, Commentary, and Recommendations

Matthew Neal
Department of Computer Science
North Carolina State University
Raleigh, North Carolina, USA
meneal@ncsu.edu

Joseph Sankar
Department of Computer Science
North Carolina State University
Raleigh, North Carolina, USA
jesankar@ncsu.edu

Alexander Sobran
Department of Computer Science
North Carolina State University
Raleigh, North Carolina, USA
aisobran@ncsu.edu

ABSTRACT

This paper provides a comprehensive review and commentary on software defect prediction research. Review and criticism of previous work is presented. Recommendations and future steps in the domain are given.

Keywords

Fault prediction model, Software mining, Ant Colony Optimization, Classification, Defect Prediction

1. INTRODUCTION

It is a well-known fact that software bugs are much cheaper and easier to fix before being released. But finding these bugs is often difficult and some bugs may be cheaper to fix than others. Researchers have been working on fault detection models to predict which software modules are most likely to contain bugs post-release. Management can use these predictions to focus testing and bug-fixing efforts on those modules, resulting in fewer bugs in release which are less costly to fix.

In this paper, we focus on the progress of software fault prediction models in the literature within the last few years. Section 2 contains an assortment of existing work in the area. We analyze the literature through different perspectives in Sections 3 through 10. Finally, in section 11, we give our overall observations and suggestions for future research in this area.

2. RELATED WORK

Cagatay Catal and Banu Diri [5] gave an overview of software fault prediction studies and advancements up to 2008. They found that an increasing number of studies used datasets available to the public. They also found that since 2005, machine learning algorithms have become increasingly popular choices to implement the models. Finally, they observed that the most dominant metrics in fault prediction were at the method level. They recommended that machine learning algorithms and public datasets continue to be used, but caution against using method-level metrics and instead suggested class-level metrics as they can predict faults earlier in the software development cycle.

A couple of years later, Hall et al [9] performed another comprehensive literature review on fault prediction performance from January 2000 to December 2010. They hoped to see how the context, independent variables, and modeling techniques affected the performance of fault prediction models. They found that simpler modeling techniques seemed to perform better, along with feature selection on sets of independent variables. They encouraged future studies to better report their context, methodology, and performance.

A couple of years after the Hall paper was published, Radjenovic et al [16] performed a literature review of metrics used in fault prediction models. The date range of papers was about double that for the Hall paper, from 1991 to 2011. The authors found that there are major differences between the metrics used, and that object-oriented and process metrics seemed to be better at predicting faults than static code metrics.

Vandecruys et al [19] mined software repositories to create predictive models. The authors used AntMiner+, an Ant Colony Optimization (ACO)-based classification technique. On public datasets, AntMiner+ was found to be competitive to alternative classification techniques, such as C4.5, logistic regression, and support vector machines. The authors suggested that software managers would find the output of rules produced by AntMiner+ easy to understand and accept.

While there have been plenty of studies about predicting software faults based on the code itself, few studies have looked at organizational structure as a factor. Nagappan et al [13] used organizational metrics, such as number of engineers, edit frequency, and organizational intersection factor to predict fault-proneness. The authors compared the effectiveness of the resulting model with alternative models which use traditional software metrics, like code churn and code complexity. They found that the model derived from organizational metrics had better precision and recall than others derived from software metrics, meaning they can also be effective indicators of failure-proneness.

Bird et al [3] examined whether development that is largely distributed produces more failures than development that is

DOI: 10.1145/1235

mainly collocated. The belief at the time was that global development was prone to more failures than collocated development. They found a negligible difference in both code metrics and failures between the two methods of development. The authors examined how the developers working on Windows Vista managed to work well together among teams located in different countries based on the relationships between the development sites, cultural barriers, communication, consistent use of tools, end to end ownership, common schedules, and organizational integration. They recommended that companies wishing to distribute development across sites located far apart to employ similar strategies to overcome some of the difficulties associated with such an endeavor.

Arisholm et al [1] examined different ways to build and evaluate fault prediction models. First, they tested a variety of modeling techniques, such as neural networks, C4.5 with some variants, support vector machines, and logistic regression. They then looked at different metrics: Process measures, object oriented code measures, and delta measures. Finally, they looked at ROC area and cost-effectiveness as evaluation criteria. They found that the choice of modeling technique did not have much of an impact when evaluated with both criteria tested. Among the metric sets, they found that process measures provide a significant improvement over the others, even though they are typically more costly to collect. They also suggested cost-effectiveness as a good evaluation technique instead of traditional techniques like precision and recall as smarter decisions can be made to prioritize which parts of the project are tested and bug-fixed.

Jun Zheng [22] argued that the mistakenly predicting a module as non-defective is more dangerous and costly than mistakenly predicting a module as defective. He presented three algorithms which boost neural networks to predict software defects with cost in mind. When evaluated on four NASA datasets, he found that threshold moving algorithm gave the best results in the sole Normalized Expected Cost of Misclassification (NECM) measure. He especially recommended threshold-moving algorithms on projects written in object oriented languages.

Most software products are structured in a hierarchical format, for example into methods, classes, files, packages, etc. What level of analysis should fault prediction models operate on? And can analysis on one level of study apply to other levels? Posnett et al [15] examines these issues. They observed that sometimes relevant phenomena only occur at an aggregated level or data may only be observed at an aggregated level, meaning that studies are often conducted at aggregated levels. They also noted that in other fields of study there are ecological fallacies which make findings at aggregated levels not apply at disaggregated levels. They found that much of these fallacies also exist in the domain of computer software and that care needs to be taken when employing ecological inference. As to how exactly the risks of ecological fallacies can be dealt with, the authors left open for future research.

Traditional software fault prediction models are trained on historical project data. Since new projects do not have such a volume of historical data, these fault prediction models are not as effective at within-project defect prediction. A possible solution to this issue is cross-project prediction which uses data from one project to predict defects in another. Still, models using this technique exhibit poor performance. Rahman et al [18] argued that one reason for such behavior is that standard evaluation measures such as precision, recall, and F-measure are taken at specific threshold settings, while they really should be taken in a range of time/cost vs. quality trade offs. They took the standard measures at a variety of tradeoffs and found that cross-project defect prediction is at least as good as within-project defect prediction, and sometimes substantially better.

Also within the area of cross-project defect prediction, Nam et al [14] found that when the source and target projects have different feature distributions, the resulting model gives poor performance. The authors use Transfer Component Analysis (TCA) to find a common feature set for both projects and then map the data of both projects to it. They found that TCA is sensitive to normalization, so they developed an improvement, TCA+ to select appropriate normalization options. After using TCA+ to create cross-project defect prediction models for eight open-source projects, they found a significant improvement in prediction performance compared to traditional algorithms and techniques. The authors proposed applying knowledge in one domain to another to further improve performance.

As another approach to improving fault prediction, Tian Jiang [11] introduced *personalized defect prediction*. He argued that developers have different coding styles and techniques and that if each developer had their own defect prediction model, performance would improve. He was careful to note that the developer was not a feature of the model, but that there was a model for each developer. He ran experiments on six large open-source projects using PCC+, a model which chooses the highest confidence prediction among CC (traditional change classification), PCC (personalized change classification), and weighted PCC (PCC with changes from other developers added to a developer's model). Compared to CC and MARS (another predictor which also creates different models for different groups of data), PCC+ outperformed CC, MARS, and PCC as long as there is enough training data for each developer. Jiang recommended that PCC+ be applied to other recommendation systems and types of predictions.

Two common approaches to evaluating defect prediction models are precision-recall and ROC curves. Davis et al [8] showed that there exists a deep connection between the two. Additionally, they claimed that ROC curves are overly optimistic with highly skewed data, and that PR curves are better suited for that type of data. While there exists a connection between the two types of curves, the authors showed that an algorithm that optimizes the area under the ROC curve does not necessarily optimize the area under the PR curve. Essentially, they were trying to show that although both evaluation methods are related, just one does not show the complete picture. Both are needed to gain a better understanding of the performance of an optimization algorithm.

Given that managers are trying to improve the process of software development, how do we best manage such a process? Chidamber and Kemerer [6] developed and evaluated six such metrics. They were then tested at two different sites via automated tools. The authors concluded by suggesting how these metrics can be adapted in a software development environment and correlated with managerial performance indicators.

D'Ambros et al [7] presented a benchmark for fault pre-

diction. Their motivation was the observation that such a benchmark did not exist, making it difficult to compare the models and approaches found in the literature. Their benchmark consisted of a publicly available data set consisting of five software systems. They also compared and evaluated well-known fault prediction methods and developed some new ones. They then ran their new methods on their benchmark, finding that they actually performed the best out of all. They conclude by stating that techniques based off of only one metric do not work well against every software project, meaning that the more useful metrics included, the better.

Kim et al [12] introduced a new technique called *Change Classification*, which looks at individual changes to source code and predicts whether those changes are similar to previous buggy or clean changes. When testing on twelve open-source systems, change classification was found to be as accurate as other leading bug prediction techniques. Since this was a new technique, there were a number of open issues for future research.

Wu et al [20] implemented an algorithm which recovers links between bugs and committed changes called *ReLink*. The authors tested this new algorithm on three open-source projects, finding that it was roughly as accurate as traditional heuristics.

Zhang et al [21] attempted to implement a universal defect prediction model, which would work on both new projects and projects with historical data. In order to account for variations in the distributions of predictors, the authors clustered projects based on the similarities of the predictors. Their dataset included almost 1,400 open-source projects. They found their universal model performs well on both within-project and cross-project models. They have hope that such a universal model is achievable.

3. ECOLOGICAL INFERENCE

Ecological inference or EI was presented in the first paper we reviewed, the Posnett paper [15]. Posnett et al explain ecological inference as the idea that an observation or finding at an aggregated level can apply at a disaggregated level. For example, the idea that a model of defects at the package level also applies at the file level.

The suggestion is that in the fault prediction area there is in fact an Ecological Fallacy, where findings at the aggregated level are expected to be valid at a disaggregated level. As in the example above a defect model that is created at a package level is applied to predict defects at a file level. Based on this concept Posnett wanted to prove whether or not the Ecological Fallacy applied to fault prediction.

Posnett et al posit two main overarching conclusions in their paper: That aggregated models when evaluated only at the aggregated level may have deceivingly strong performance, and furthermore inferences from aggregated models may not apply to their disaggregated parts [15]. Posnett cites another of our papers, Bird et al, as a potential rebuttal of his results because of the fact that geographical distribution did not have an effect in Bird's study [3].

Moving beyond the Posnett paper, Rahman et al in 2012 made considerations regarding the aggregation level of their models, in that they decided to examine code at the file level. Rahman et al went that direction because they were worried that evaluation of a model on a coarser level of aggregation may not be a good predictor at the file level [18]. Rahman again cites Ecological Inference in a 2013 paper on Process Metrics as a threat for validity [17].

3.1 Discussion

The main question is whether EI and EF represent important concepts to researchers going forward in the fault prediction area. It seems that only a subset of papers are actually being written currently where EF is recognized as a concern. Our recommendation would be further study to validate Posnett's 2011 work, and if the work is further validated the community should consider EI/EF when conducting studies.

4. MACHINE LEARNING ALGORITHMS

Every fault prediction study leverages a machine learning algorithm as the heart of the model. Machine learning algorithms are a way to "learn" from data and create a model that can make predictions. Typically, there is a set of training data which the algorithm uses to generate the model. Then, it is evaluated with test data to see if it generates accurate and reliable predictions. There are a variety of ML algorithms including regression, neural networks, support vector machines, and decision trees.

4.1 Regression

Regression is a ML algorithm for data that can be fitted to a curve. Regression algorithms are considered to be the simplest type, but can perform well if the data is a well-fit to some sort of well-behaved mathematical function. Most of the referenced papers used some form of regression as their main algorithm ([15], [3], [13], [18]).

4.2 MARS

MARS is short for Multivariate Adaptive Regression Splines. The concept goes back to a 1991 paper by Friedman the technique is a non-parametric method used to model complex multivariate non-linear scenarios. It is suited modeling non-linear data because of the fact that models have a series of hinge and basis functions. Each hinge function per Bettenburg et al "partitions the data into disjoint regions that can be described separately"[2]. Bettenburg et al used MARS in building models to look at the difference between global and local trained model. The Jiang paper used MARS models based on Bettenburg's work as a baseline to compare PCC and PCC+ models against [11].

4.3 Neural Networks

Neural networks are a family of machine learning algorithms based initially on the perceptron. The basic structure of neural network consist of an input layer, layers of activation functions, and an output layer. Each layer is connected to other layers but no connections exist within the layer. The result of each activation function is delivered to the next connected layer until it finally reaches an output layer.

Jun Zheng [22] used a back propagation neural network (BPNN) with AdaBoost, as a way to achieve better classification results by producing diverse base classifiers. Zheng states that he used neural networks based on their popularity in pattern recognition, and BPNNs because they were the most frequent neural network used in the literature. AdaBoost was chosen because it is a popular way to improve the performance of the model.

4.4 Naive Bayes

Zhang et al [21] use Naive Bayes classifier as a modeling technique in their approach to build a universal defect prediction model for cross-project defect prediction. They decided on this classifier because previous research determined that Bayes learners outperform other learners when predicting defects in noisy data. The Naive Bayes classifier is based on Bayes' theorem with independence assumptions.

4.5 Probabilistic Graphical Models

Probabilistic graphical models are considered powerful machine learning tools. They consist of a graphical structure where nodes are events and edges are conditional dependencies. Some examples include bayesian networks and hidden markov models. They have many strength including informative visualization from structures, insights from model inference, and lower complexities. Inferring insights from model structure could be leveraged in software defect prediction. Certain conditional relationships can be reported verbally to all concerned parties. It has also been posited every machine learning algorithm is an instance of a probabilistic graphical model.[4]

4.6 Other ML Algorithms

Vandecruys et al [19] used a novel algorithm known as AntMiner+, which is based on Ant Colony Optimization(ACO) which models the foraging behavior of ant colonies. In AntMiner+, ACO is used to optimize a rule based classification algorithm. Rule based classification algorithms usually have greater comprehensibility than other machine learning algorithms as the association rules can be reported. Vandecruys also presented good accuracy using AntMiner+.

Tian Jiang's algorithm [11] was based on a classification scheme known as Personalized Change Classification+ (PCC+).

4.7 Discussion

Catal et al [5] suggested that more models be based on machine learning techniques rather than statistical methods or expert based systems. We see that research has followed the advice of the authors. Among them, Arisholm et al [1] did not find any major differences among the different machine learning techniques, although they found that C4.5 with Adaboost gave the best results.

Not only have authors used well-known ML algorithms, but they have even implemented new ones. This shows that ML algorithms were the primary choice of recent studies and still are today.

5. EVALUATION METRICS

There are three major approaches to evaluating the performance of models in the area of fault prediction: Confusion matrix approaches, ROC curve based approaches, and finally cost effectiveness approaches.

5.1 Confusion Matrix Approaches

The use of confusion matrices, and the metrics (precision, recall, and F-measure) derived from them are very common in the fault prediction area. As will likely be known by the reader, a confusion matrix is a contingency table summarizing the predicted values compared with actual values. A confusion matrix in most cases will have been created based on test data where the actual values are known. Precision, recall, type I errors, type II errors, and F-Measure are all statistics that can be used to assess how close a model has come in predicting the actual on the test data. This methodology is used in nearly all of the papers we read [15] [14] [3] [19] [13] [1] [18] [11] [21]

Even though the methodology is fairly standard throughout the papers we read, what actual metric is reported between precision, recall, overall accuracy, type I errors, type II errors, and F-Measure has considerable variance between papers. The Vandecruys paper [19] even uses terminology that is separate from many other publications in using sensitivity for recall and specificity for the true negative rate, which is a seldom reported statistic. One suggestion in [9] is to present the confusion matrix itself rather than to present one statistic or the other when possible (where possible is determined by the sheer number of confusion matrices that would be produced). The idea is that it would better enable comparison across studies since researchers could produce whatever statistic they happen to need from the data itself.

Another measurement we encountered is the g-measure. It is used by Zhang et al [21] in their cross-project defect prediction study. They use it to combat the the instability of precision in datasets with low percentages of defects.

5.2 ROC curves

The ROC curve is another fairly standard metric for performance used in the literature. ROC, or Receiver Operating Characteristic analysis is an elaboration of the confusion matrix approach. The idea is to use the rates of false positives and true positives on an x-y plane and to have x values represent the rates of false positives, and y values represent the rates of true positives. The rate for a particular model can be plotted as a point on the ROC curve and the area under the curve (AUC) be used as a metric to determine the quality of the model with a point at (0,1) being optimal [15].

These methods are used in a few of the papers we reviewed [18] [1] [15]. Some negatives have been pointed out about the ROC approach and its usefulness in certain circumstances. One such negative is the fact that ROC curve approaches have "an overly optimistic view of an algorithm's performance if there is a large skew in the class distribution". [8] Other papers mention the fact that both ROC/AUC approaches and confusion matrix based approaches in general do not address the whether a particular fault prediction methodology can pinpoint the source of the faults or not. Since knowing where to find errors is just as important or more important than knowing whether errors exist or not several papers find ROC/AUC insufficient as an evaluation metric [15] [1] [18].

5.3 Cost Effectiveness

Arisholm et al [1] contributed Cost Effectiveness as a metric to software defect prediction. The high level idea behind Cost Effectiveness is the idea that it is impossible in most cases to inspect the entire code base of a large project for bugs. Defect prediction models that can guide inspection of code to find the largest number of bugs by inspecting the smallest percentage of the code base are viewed as models with high cost effectiveness.

The basic implementation of the metric is a set of two curves on an x-y plane that has a percentage of faults as the y-axis, and the percentage of LOC included in classes se-

lected to focus verification as the x-axis. Classes are ranked according to their likelihood of having defects, first by the model, and next by the size of the class in case of ties. The graph has a baseline of $y = x$ which is essentially the assumption that the percentage of faults will be equivalent to the percentage of lines of code inspected, this is done using a random ranking of the classes. On the same graph is the cost effectiveness curve which is the actual percentage of faults given the percentage of LOC of classes selected to focus verification according to the ranking determined by the model.

Arisholm et al use an area calculation that is normalized to be a proportion of the optimal area under the curve. To find an approximation of the optimal cost effectiveness they use a ranking that ranks the most error filled classes highly and then create a cost effectiveness line according to that ranking. The actual calculation is
$CE_\pi = (CE_{pi}(model) - CE_{pi}(baseline))/$
$(CE_{pi}(optimal) - CE_{pi}(baseline))$ Other papers have used a simpler calculation of the area under the cost effectiveness curve, or AUCCE [15]. The same concept is also used by Rahman et al under the name AUCEC [18].

Those same papers that find ROC to be inefficient all suggest that Cost Effectiveness is an important and to some degree a superior metric to use in determining the performance of an algorithm [15] [1] [18]. The Arisholm paper was in fact one of the most influential papers that we worked with throughout the semester. Cross-project defect prediction has been validated to some degree by the existence of Cost Effectiveness as an evaluation metric [18]. The technique was used in Jiang et al as a metric as well [11].

5.4 Discussion

It seems that Arisholm's contribution to the community with Cost Effectiveness back in 2010 has provided a better metric to evaluate fault prediciton models going forward. Seeing as the point of fault prediction should be to build tools that can be used in the real world for real projects, injecting the realism that CE brings to the table seems to be totally appropriate in going forward. Even though that is viewed as the case by some in the community, confusion matrix based approaches and the related ROC curve concept are seemingly going nowhere. These approaches will continue to be important in evaluating models until something better comes along.

Our recommendation in this area would be to work with a hybrid approach of using both confusion matrix approaches and the ROC curve as well as CE based metrics. The suggestion of Hall as listed above to simply report confusion matrices seems like a great recommendation when possible going forward. At that point researchers could simply report cost effectiveness data along with confusion matrices and leave any further analysis to those that are interested. If reporting confusion matrices were not to become popular, settling on one confusion matrix based approach such as the F-Measure would be preferable to the random mixture of reporting approaches viewed throughout our papers.

6. FEATURES

This section could also be labeled metrics, as it is in many places throughout the literature on this topic, but to not confuse it and the previous section on evaluation metrics we have chosen to label it features. The main reasoning for this is the fact that metrics are actually used as features in Machine Learning algorithms and not actually as metrics for comparison of one method versus another. The Radjenovic paper [16] is a fairly exhaustive paper with relation to metrics/features in fault prediction. Radjenovic came up with three metric categories to discuss in their literature review, and we will stick with those same categories in relation to the papers we reviewed. The categories are: Traditional, Object-Oriented, and Process. Some of the papers we reviewed defy this categorization and use metrics that are either one offs or are hybridized to such a degree that they should be explained on their own.

6.1 Traditional

These are metrics that relate to complexity (ie McCabe or Halstead calculations) and lines of code. A good number of papers we reviewed used these metrics as features [15] [19] [3]. There is considerable discussion in terms of whether these sorts of metrics provide good quality models or not. Hall et al indicate that in their review that models using only static code metrics that are complexity based have poor performance, but that Lines of Code based models tend to be useful [9]. Radjenovic et al indicate that LOC based methods have no strong evidence behind them in terms of faults furthermore they state that smaller studies in general have given more weight to LOC and size metrics than larger studies, because of that they rated the effectiveness of the metric as "moderate"[16]. On complexity metrics Radjenovic states that these metrics are reasonable, but that "others are better"[16].

6.2 Object Oriented Metrics

Object oriented metrics seem to have been dominated by the metrics presented by Chidamber and Kemerer [6] [7]. These metrics include things like number of children(NOC), coupling between classes (CBO), or lack of cohesion of methods (LCOM). Object oriented metrics were used in only one of our papers, Arisholm et al, and used there really only as a comparison metric[1]. In terms of evaluation of their success or failure Radjenovic et al indicate that OO metrics are useful but that there is some debate about whether OO metrics and size metrics are correlated and that further work is needed in that area [16]. Hall et al state that OO and LOC are actually on pretty equal footing and that OO provides better performance than source code metrics [9].

6.3 Process Metrics

Process metrics relate to things like number of developers, code churn, number of commits, features, etc. Process metrics are mostly mined from source code management systems like Git, and also bug tracking and issue tracking systems like Jira. These metrics have a strong following in the fault prediction community and are well represented in our papers [15] [3] [1] [18]. Rahman et al particularly only used a set of process metrics while the other papers mentioned used some sort of combination of process and other metrics.

There is some dispute between Radjenovic and Hall with regards to process metrics, with Hall suggesting that their performance was the worst out of everything they had examined where Radjenovic considers process metrics to be promising and that they are deserving of further study and validation while additionally stating that they provide superior post-release fault prediction [9] [16].

Another discussion of the strength of process metrics can be found in a 2013 paper by Rahman where he and Devanbu examine "How, and Why, Process Metrics are better"[17]. Based on the title it is fairly clear where Rahman and Devanbu come out on this, but the main thrust of their argument is that process metrics do not suffer from the stasis that code metrics do suffer from. The idea is that between releases code metric based models lose relevancy as the distribution of defects in the project changes. Further Rahman indicates that this assessment is geared towards situations in which AUC and Cost-Sensitive evaluation methods are being used.

Process metrics were also used by Zhang et al [21] to build a successful universal defect prediction model. This reaffirms the feature set as generalizable to other projects when performing cross-project defect prediction.

6.4 Organizational Structure

This is an approach that was proposed by Nagappan et al at Microsoft Research [13]. The metrics included a set of 8 measures of organizational complexity: Number of engineers, Edit Frequency, Depth of Master Ownership, Percentage of Org contributing to development, Overall Organization Ownership, and Organization Intersection Factor. They used these metrics to create a fault prediction model and compared its performance in Precision and recall against a set of OO, Traditional, and Process metrics. For Windows Vista they show that Organizational metrics are superior to the other types of metrics available. As mentioned in the paper there are questions about whether this approach would work outside Microsoft or with a smaller project. Naggapan and Bird produced another paper in the next year using organizational metrics again with a bit of a twist in looking at whether distributed development had an effect on software quality [3]. This paper added an additional metric of the actual location of the members of teams working on Windows Vista with levels starting at the same building and growing to the point of developers working at locations across the globe from one another. Organizaitonal metrics as well as a set of process and traditional metrics were used in conjunction with the distribution metric seemingly out of surprise that there was little difference between distributed projects and more local projects. The idea was to try to prove that there was not in fact some correlation between the other metrics they looked at and the performance of teams across long distances. They found that in fact there was no difference and that distributed development within one company was possible without sacrificing quality.

6.5 Text Classification

The paper on Personalized Defect Prediction by Jiang et al uses a methodology based on change classification [11]. To understand the metrics involved it was necessary to go back to a paper by Kim et al and get an idea of exactly what was done for feature extraction [12]. The basic methodology uses process metrics such as changed LOC between commits, and when commits take place, but also uses traditional techniques like LOC, and complexity analysis.

The more interesting and complex aspect of both of these papers is text classification. They both use a tool called BOW (Bag Of Words) to pull features out of log messages (called metadata in Jiang et all) and source code. The metadata aspect of this approach is also used in the Arisholm paper under the name Deltas where it is stated as simply the change between releases and also calls it Code Churn [1]. Code churn in later papers is simply subsumed into process metrics, where later survey papers like Radjenovic simply categorize them together [16].

The source code inspection can be incredibly low level as in the Kim paper where they "This (text classification) means that every variable, method name, function name, keyword, comment word, and operator, that is everything in the source code separated by whitespace or a semicolon is used as a feature"[12]. The Jiang paper pulled out features like incorrect calls of individual language level functions like malloc and calloc on a per developer basis and then used those to predict faults for individual developers. It is an intriguing approach and one that was successful based on Jiang's results, It is an approach that has been cited, but not repeated. Partially that may be because the paper is fairly recent having been published in 2013, but it may also owe something to the fact that this is a seemingly intensive effort that few would use in industry.

6.6 Benchmark Data Sets

In one of our most recent papers (the Nam paper 2013 [14]) we found researchers using benchmark data sets that had sets of metrics that could be chosen from in order to build models. The two sets discussed in the paper are ReLink and AEEEM [20] [7].

D'Ambros' AEEEM seemingly took everything he could find that he could rationalize and some new ideas and then just threw all of it into a metric set. AEEEM includes: change metrics, CK metrics, Object Oriented metrics, number of previous defects, complexity of code change, churn of CK and OO metrics, entropy of CK and OO metrics. ReLink unfortunately was not as helpful in listing out what is available in terms of metrics. Based on the Nam paper it contains 26 complexity metrics [14].

Rahman et al also use a benchmark set of metrics for their 2013 paper. The metric set is called UNDERSTAND from Scitools. It uses a set of complexity, LOC, and OO metrics [17].

6.7 Discussion

As can be seen by the length and complexity of this section, the metrics area is still a very open area in fault defect prediction. The simple fact that there are major differences in terms of what metric set is being used across different projects creates some difficulty when comparing them. Also since many of these projects also use a hybrid approach, it makes it even less likely that another project will use exactly the same set of metrics.

There is simply no consensus on what a good metric set really is, and that makes research going forward a difficult proposition. Our recommendation would be for the community to continue to codify what they believe to be the best metric sets and then to build benchmark data sets like AEEEM and ReLink that can be used across projects so that the approaches of researchers can be appropriately compared.

7. DATA SOURCES

The choice of dataset is extremely important, especially when doing software defect prediction research. Datasets should be representative of actual software projects and should

ideally be accessible to the public so the results can be retested and verified by others. Within the referenced papers, we found a variety of data sources which we feel need to be discussed. Primarily, we found open-source datasets, NASA datasets, and Windows Vista datasets. We will discuss each along with some reasons why they may have been chosen.

7.1 Open-source

The majority of papers we reviewed used open-source data, with most of those being Apache projects. Open-source projects are accessible to everyone, so researchers can verify the results they read, making every author accountable for the results they publish. Many of the Apache projects are tracked using JIRA which contains a plethora of data, including defect information.

Apache libraries are also commonly used in a variety of products, so defects that are in released code can manifest themselves in projects that use them, creating a sort of ripple effect.

Sourceforge and GoogleCode projects were used by Zhang et al [21] for cross-project defect prediction. Although open source projects are a viable data source they also contain many useless projects where defects are not properly tracked.

7.2 NASA

A couple of papers ([22], [19]) used datasets from NASA. Among them were KC1, used for storage management, KC2, used for scientific data processing, CM1, used for NASA spacecraft instruments, and PC1 and PS4, used for flight software.

One major reason for using NASA datasets is that most of their software is critical. A malfunction or crash due to a software defect is not only costly, but can also be deadly. NASA's software needs to be thoroughly tested and bug-fixed before it is ready for use. Software defect models which work well on these datasets can be claimed to be more reliable and trustworthy. Additionally, NASA's datasets contain code metrics and defect information, which is easy to mine and create training and test data from.

NASA datasets are publicly available. Researchers can more easily verify the results obtained in a study. Also, since these datasets are static, they can be used as a baseline to compare the results from a newly created model against those from other studies.

7.3 Windows Vista

Two of the referenced papers (Bird et al [3] and Nagappan et al [13]) use data from Windows Vista. Neither paper tests a fault prediction model using Vista test data. Instead, the first paper examined how different development teams worked together within Microsoft and the second paper examined Microsoft's organizational structure. Two of the three authors of the second paper are associated with Microsoft Research.

But why was Windows Vista chosen? Windows in general is a widely used operating system which contains many lines of code. Windows is also known for being vulnerable without frequent patching and antivirus software. Windows Vista may have been the latest version of Windows at the time of both papers, but Vista is also known as a slow and buggy release. Maybe the authors intended to show that Vista did not contain an unusually high number of bugs compared to other releases, or maybe they just wanted to choose a Microsoft product that was well-known, whether for good or bad.

7.4 Discussion

Most of the studies we examined used open-source datasets which is in line with previous recommendations ([5], [19]). The two papers which used Windows Vista were not proposing a new or improved fault prediction model but were instead analyzing Microsoft's organizational structure to see how it may have affected bug levels. Although the NASA software isn't open source, we think that its datasets are acceptable because of their domain and the fact that NASA has made the datasets public for research purposes. Research has heeded the recommendations of earlier researchers and now primarily use open-source datasets

8. DATA PREPROCESSING

8.1 Filtering

Zhang et al [21] filtered for projects to meet specific programming language criteria. To perform this they specifically filter for file types associated to those languages. They also apply a second filtering to remove small projects by filtering out projects with a small number of commits. The decide on the 25% quantile which is 32 commits. This is to confirm projects have enough information for extracting viable features. Next they filter projects with a lifespan of less than one year and project with limited defects for the same reasoning. Finally, they filter out projects without fix-inducing commits as they are marked abnormal.

Vandecruys et al also utilize a filtering based approach to rank variables according to their predictive power. They used a χ^2 based filter. The idea was to decrease the number of metrics to the point that only the metrics that were useful would be actually added to the model [19].

8.2 Discretization

Generally a method to turn continuous variables into discrete variables. Vandecruys et al used this technique because AntMiner plus could not deal with continuous variables [19].

8.3 Oversampling

The data in many fault prediction data sets has many more clean files than faulty files the data is considered to be heavily skewed. One method for dealing with skewed data is oversampling, as discussed in [19]. The idea is when sampling for training data to repeat observations that correspond to faults in the data. Vandecruys et al found in their study that oversampling had different effects at different levels of oversampling (the number of times faulty observations were repeated).

The true positive rate, sensitivity or recall, and the true negative rate, specificity are affected by the levels of oversampling. In Vandercruy's case an increase in oversampling increased specificity, but decreased sensitivity. This change also coincides with a decrease in accuracy. Their perspective was that it was important to reach a break even point in terms of oversampling where they could live with the level of accuracy and sensitivity on one hand but specificity on the other. They found that oversampling to the level where there were equal numbers of faulty and non-faulty observations was optimal[19].

8.4 Normalization

Normalization was a key part of Nam et al's paper on TCA. TCA requires normalization to function correctly, this is fairly common throughout that statistical/machine learning/ASE world, and is essentially scaling all features in a data set to the same scale. This is often necessary because features may well be from completely different scales such as liters of oil and pounds sterling.

Nam et al use 4 different schemes for normalization: The first is a simple Min-Max scheme where $norm_x = \frac{x-min}{max-min}$. The other methods are z score normalizations with slight changes, such as $norm_x = \frac{x-mean}{std_dev}$. The difference comes in where the mean is either the mean of all of the features, or the mean of a particular feature vector. As follows in the section on TCA+ Nam et al used this as an optimization for TCA since it had such a high sensitivity to normalization[14]

8.5 PCA

PCA or principal component analysis does not completely fit into data preprocessing, but is not significant enough to be its own section. The technique is used to take a large number of features and flatten them into a smaller number of features through a transformation using Eigenvectors and Eigenvalues [10].

This technique is used by Nagappan et al, they actually use PCA in a somewhat odd manner. They first state that there is a concern with multicollinerity between the organizational metrics that the propose, so they decide to use PCA to deal with the multicollinerity in the data. They find that PCA returns 8 principal components and then determine on that basis as well as step-wise regression that all of the measures are valuable and should be retained [13]. It is possible that this is a legitimate use for PCA, but it is not a use that we are familiar with.

9. STATISTICAL TESTS

9.1 Mann-Whitney U test

The Mann-Whitney U test, a non-parametric test that determines if two independent distributions have equally large value, was used by Zhang et al [21] to pair-wise compare the distribution of metrics of projects in the domain of cross-project defect prediction.

The Mann-Whitney test is also used in Bird et al to test the differences in mean number of failures between projects that were distributed and non-distributed projects. [3] Bird et al indicate that the usage of the Mann-Whitney test is more appropriate to their circumstances because of the fact that the number of failures that they were looking at were not normally distributed.

9.2 Cliff's δ

Cliffs δ measures the effect size of the importance of the difference between two distributions. It is a nonparametric statistical test which represents the overlap two distributions. If two distributions are identical then the measurement is 0. It was used by Zhang et al [21] to determine the difference in the distributions of two project's metrics. If the difference of the effect size was large then the projects were not clustered together. This was done as second step in clustering similar projects and only on project's whose Mann-Whitney U test determined a statistically significant difference.

9.3 Wilcoxon signed-rank test

This is a hypothesis testing technique that is used in the comparison of two samples. It is a non-parametric method so, does not assume any particular distribution which increases its usefulness in certain circumstances. The Rahman paper uses the WSR test for hypothesis testing to try to prove that cross-project%SLOC AUCEC (Area under the cost effectiveness curve) is "resilient to parameter selection for a reasonable cutoff choice"[18]. This is in the context of determining the size of model in terms of number of parameters to use.

Wilcoxon Signed rank was also used in the Arisholm paper to determine whether there is a significant difference between all pairs of techniques and metric sets that they studied in their project[1]. Arisholm et al only report p-values, but also report what level of significance they used in determining the p-values. For whatever reason Rahman neglects to do so, from our perspective this is at least somewhat of a failure on the part of Rahman et al. Without the significance level it is difficult to interpret the p-values quoted.

WSR was also used in Nam et al to determine whether there was a statistically significant difference in performance between TCA and TCA+ [14]. Jiang et al used WSR to examine the difference between CC and PCC within their paper [11].

9.4 Wilcoxon rank-sum test

The Wilcoxon rank-sum test is a non-parametric statistical test. It compares the ranks of two independent population based on their whole ordered rank. Zhang et al [21] used the test to compare their technique of rank transformation of features with log transformation of features. With this measure they were able to differentiate between the two transformations.

9.5 Cohen's d

Cohen's d is a measure of effect size, per Wikipedia it is defined as the difference between two means divided by a standard deviation for the data. Arisholm et al use Cohen's D to report the effect size of the differences generated by the Wilcoxon Signed Rank test commented on above [1]. Between the two measures Arisholm creates a fantastic table presented in their paper that shows both the p-values generated and the effect sizes. Zhang et al [21] map Cliff's δ to Cohen's d standards to determine effect size of project metric distribution difference.

9.6 Spearman's Rank Correlation

This technique was used in the Nagappan paper to determine the correlation between organizational metrics. The paper suggests that Spearman's rank correlation is superior to the Pearson correlation because it does not require a linear relationship between elements under consideration [13]. Nagappan et al use the rank correlation in terms of a hypothesis test between the organization metrics that are discussed in the metrics section above. They report p-values along with the associated significance level.

Bird et al used the Spearman Rank Correlation in their paper as well, perhaps because Nagappan was the second writer on this paper as well [3]. In this circumstance they

wanted to determine whether there was a correlation between the distribution level of projects and process metrics. The reasoning was that they wanted to prove that the results they had found in terms of distributed projects being no more fault prone than non-distributed projects. If they could show that there was no real correlation between other factors and the fact that distribution was not a cause for faults.

9.7 AIC

The AIC is the Akaike Information Criterion. It is generally used as a methodology to reduce the number of features in a machine learning model. In a feature selection scenario, features are removed/added and the AIC value is checked to determine whether removing/adding a feature leads to a better model [10]. Rahman et al use AIC for the same purpose of reducing the number of variables in their models [18].

9.8 VIF

The Variance Inflation Factor is a metric used to determine whether multicollinearity exists within a particular model [10]. It was used alongside the AIC score in the Posnett paper we reviewed. As Posnett et al indicate AIC should not be used without using VIF because of the potential for high Multicollinearity. In their feature selection process they used VIF as another factor to determine which models were best. They ranked by AIC and then additionally removed any models with high VIF from consideration.

9.9 F Statistic

The F statistic is generally used to test the relationship between the independent and dependent variables in regression scenarios [10]. The Bird paper makes extensive use of the F-Statistic to determine whether the distribution level of developers has an effect on the response variable, the number of faults. The paper presents p-values with significance levels as should be standard. The p-values test the hypothesis that there is no effect of the independent variable on the dependent variables discussed above [3].

Bird et al use this setup to prove that a majority of the difference in defects is in fact the number of developers working on a binary. He attempts to prove that using the F statistic and its concurrent adjusted R^2 statistic when adding in the number of developers to the project. Since Adjusted R^2 is meant to show the amount of variance in the response variable explained by the predictor variable adding the number of developers to the model should increase the R^2 value which it does.

9.10 Discussion

Our main recommendation with regards to statical testing is that there should be at least basic significance testing in general in the defect prediction community. Surprisingly there were no statistical significance tests of any sort in one of the papers we reviewed [19]. Furthermore the reporting of p-values without reporting the statistical significance level that they were evaluated should not happen at this level.

10. CROSS-PROJECT DEFECT PREDICTION

One of the main limitations of defect prediction models is the within-project requirement for model fitting. Cross-project defect prediction can be used to predict defects in projects with models trained on other projects. This allows the application of defect predictors to new projects without enough history for within-project model fitting. It also allows the development of a universal model for defect prediction on any project.

10.1 Context Factors

Context factors for projects were assessed by Zhang et al [21] and used as a measurement of project similarity. These context factors showed an overall increase in the confusion matrix measure when compared to just code and process metrics. Context factors provide a set of measurements to rank a project's relatedness when performing cross-project defect prediction.

Programming Languages Projects can be divided into groups depending on the programming language they are written in. Projects written in multiple languages could be problematic unless multiple language categories are included.

Issue Tracking Whether or not a project uses an issue tracking system.

Total Lines Of Code Project size based on the total number of lines of code split by quartiles.

Total Number of Commits Project size as a function of the number of commits. Split into quartiles.

Total Number of Developers Project size as a determined by the number of developer. Split into quartiles.

10.2 Context-Aware Rank Transformation

Zhang et al [21] propose a context-aware rank transformation approach. First, split the project up into non-overlapped groups based on the previous context factors. Then cluster the groups with similar context factor distributions. Obtain a ranking function based using the 10th quantiles of predictors. Apply this ranking function to bin the predictor values into the 10 levels. The ranking function achieves the purpose of making projects with different scales of metrics comparable. This is crucial to a cross project defect prediction algorithm.

10.3 Clustering

Zhang et al [21] use a clustering method to group similar projects together for cross-project defect prediction. Projects in the same group are used to predict each others' defects and form a universal defect prediction model. The clustering is a two step process of comparing the distribution of the project metrics and then quantifying the difference between the distributions. The first step leverages the Mann-Whitney U test. The second step leverages the Cliff's δ measure.

10.4 TCA

Transfer component analysis(TCA) finds the latent feature space for two different projects by minimizing the distance between their distribution while preserving the original data. Features for both projects are then mapped to this latent space. Training and prediction is done on the transformed space. TCA is sensitive the normalization which decreases its performance in transfer defect prediction. Nam et al [14] extended TCA to TCA+ to deal with this sensitivity to normalization.

10.5 TCA+

TCA+ is an extension of TCA proposed by Nam et al [14]. By optimizing the normalization stage of data processing they are able to mitigate the sensitivities of TCA to normalization and demonstrate improved software defect prediction performance.

10.6 Discussion

Our recommendation for future work includes extending the universal model used by Zhang et al [21] with the TCA+ approach of Nam et al [14]. They both offer different approaches to linking projects with one relying on statistical similarities while the other relies on component analysis, respectively. A method leveraging these two approaches could first cluster based on the rank transformation and then preprocess data using TCA+ before applying a machine learning algorithm for defect prediction.

11. CONCLUSIONS

Software defect prediction has made significant progress within the last few years. Recommendations from earlier papers, such as more machine learning algorithms [5], better metrics [1] and publicly available datasets ([5], [19]) have largely been followed by the papers we reviewed.

However, there remains room for improvement for future studies, namely with ecological inference concerns [15], feature metrics [9], and statistical tests. Recent research into cross-project fault defect prediction has proved to be an area with potential since the application of cost effectiveness as a metric, we hope to see additional effort in this area. We suggest that researchers focus on these areas to continue the progress that has been occurring recently.

We recommend usage of a greater diversity of machine learning algorithms including probabilistic graphical models. A specific use case could include hidden Markov models. It could be theorized changes to a software component can be viewed as a series of changes with a hidden state of defective or not defective. This sets up a feasible implementation of software defect prediction in a hidden Markov chain where the software metrics are observable and the software defect truth is hidden [4].

12. REFERENCES

[1] E. Arisholm, L. C. Briand, and E. B. Johannessen. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *J. Syst. Softw.*, 83(1):2–17, Jan. 2010.

[2] N. Bettenburg, M. Nagappan, and A. E. Hassan. Think locally, act globally: Improving defect and effort prediction models. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, MSR '12, pages 60–69, Piscataway, NJ, USA, 2012. IEEE Press.

[3] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy. Does distributed development affect software quality?: An empirical case study of windows vista. *Commun. ACM*, 52(8):85–93, Aug. 2009.

[4] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[5] C. Catal and B. Diri. A systematic review of software fault prediction studies. *Expert Systems with Applications*, 36(4):7346 – 7354, 2009.

[6] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, 20(6):476–493, June 1994.

[7] M. D'Ambros, M. Lanza, and R. Robbes. An extensive comparison of bug prediction approaches. In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pages 31–41, May 2010.

[8] J. Davis and M. Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 233–240, New York, NY, USA, 2006. ACM.

[9] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A systematic literature review on fault prediction performance in software engineering. *Software Engineering, IEEE Transactions on*, 38(6):1276–1304, Nov 2012.

[10] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014.

[11] T. Jiang, L. Tan, and S. Kim. Personalized defect prediction. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, pages 279–289, Nov 2013.

[12] S. Kim, E. Whitehead, and Y. Zhang. Classifying software changes: Clean or buggy? *Software Engineering, IEEE Transactions on*, 34(2):181–196, March 2008.

[13] N. Nagappan, B. Murphy, and V. Basili. The influence of organizational structure on software quality: An empirical case study. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, pages 521–530, New York, NY, USA, 2008. ACM.

[14] J. Nam, S. J. Pan, and S. Kim. Transfer defect learning. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 382–391, Piscataway, NJ, USA, 2013. IEEE Press.

[15] D. Posnett, V. Filkov, and P. Devanbu. Ecological inference in empirical software engineering. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, ASE '11, pages 362–371, Washington, DC, USA, 2011. IEEE Computer Society.

[16] D. Radjenovic, M. Hericko, R. Torkar, and A. Zivkovic. Software fault prediction metrics: A systematic literature review. *Information and Software Technology*, 55(8):1397 – 1418, 2013.

[17] F. Rahman and P. Devanbu. How, and why, process metrics are better. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 432–441, Piscataway, NJ, USA, 2013. IEEE Press.

[18] F. Rahman, D. Posnett, and P. Devanbu. Recalling the "imprecision" of cross-project defect prediction. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, pages 61:1–61:11, New York, NY, USA, 2012. ACM.

[19] O. Vandecruys, D. Martens, B. Baesens, C. Mues,

M. De Backer, and R. Haesen. Mining software repositories for comprehensible software fault prediction models. *J. Syst. Softw.*, 81(5):823–839, May 2008.

[20] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung. Relink: Recovering links between bugs and changes. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, pages 15–25, New York, NY, USA, 2011. ACM.

[21] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou. Towards building a universal defect prediction model. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 182–191, New York, NY, USA, 2014. ACM.

[22] J. Zheng. Cost-sensitive boosting neural networks for software defect prediction. *Expert Systems with Applications*, 37(6):4537–4543, 2010.