Expert Systems with Applications 36 (2009) 7346–7354

Contents lists available at ScienceDirect

Expert Systems with Applications

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

Review

A systematic review of software fault prediction studies

Cagatay Catal [a,*], Banu Diri [b]

[a] The Scientific and Technological Research Council of Turkey, Marmara Research Center, Information Technologies Institute, Kocaeli, Turkey
[b] Yildiz Technical University, Department of Computer Engineering, Istanbul, Turkey

ARTICLE INFO

Keywords:
Machine learning
Automated fault prediction models
Public datasets
Method-level metrics
Expert systems

ABSTRACT

This paper provides a systematic review of previous software fault prediction studies with a specific focus on metrics, methods, and datasets. The review uses 74 software fault prediction papers in 11 journals and several conference proceedings. According to the review results, the usage percentage of public datasets increased significantly and the usage percentage of machine learning algorithms increased slightly since 2005. In addition, method-level metrics are still the most dominant metrics in fault prediction research area and machine learning algorithms are still the most popular methods for fault prediction. Researchers working on software fault prediction area should continue to use public datasets and machine learning algorithms to build better fault predictors. The usage percentage of class-level is beyond acceptable levels and they should be used much more than they are now in order to predict the faults earlier in design phase of software life cycle.

© 2008 Elsevier Ltd. All rights reserved.

1. Introduction

This paper reviews several journal articles and conference papers on software fault prediction to evaluate the progress and direct future research on this software engineering problem. Many researchers used different approaches such as genetic programming (Evett, Khoshgoftaar, Chien, & Allen, 1998), neural networks (Thwin & Quah, 2003), case-based reasoning (El Emam, Benlarbi, Goel, & Rai, 2001), fuzzy logic (Yuan, Khoshgoftaar, Allen, & Ganesan, 2000), Dempster–Shafer networks (Guo, Cukic, & Singh, 2003), decision trees (Khoshgoftaar & Seliya, 2002), Naïve Bayes (Menzies, Greenwald, & Frank, 2007), and logistic regression (Denaro, Pezzè, & Morasca, 2003; Schneidewind, 2001) to predict software faults before testing process. We applied Artificial Immune Systems paradigm for fault prediction during our Fault Prediction Research Program (Catal & Diri, 2007a, 2007b, 2008).

This review does not describe all these prediction models for practitioners in detail. Our aim is to classify studies with respect to metrics, methods, and datasets that have been used in these prediction papers. We evaluated papers published before and after 2005 with respect to metrics, methods, and datasets because PROMISE repository has been created in 2005. PROMISE repository includes a collection of public datasets to build repeatable, refutable and verifiable models of software engineering and it was inspired by UCI Machine Learning Repository which is widely used by researchers in Machine Learning area (Sayyad & Menzies, 2005).

Jorgensen and Shepperd (2007) provided a systematic review of software development cost estimation studies and our review methodology is similar to their methodology. According to our knowledge, this is the first study which provides a systematic review of software fault prediction studies from different perspectives. We posed the eight research questions shown in Table 1 and these questions helped us to collect the necessary information from papers in our review process.

This paper is organized as follows: Section 2 describes the review process. Section 3 reports the results. Section 4 suggests issues for future research on software fault prediction.

2. Review process

2.1. Inclusion criteria

We included papers in our review if the paper describes research on software fault prediction and software quality prediction. We excluded position proceedings and papers which do not include experimental results. Papers with respect to their years, datasets, metrics, techniques, evaluation criteria and results have been examined. The inclusion of papers was based on the similarity degree of the study with fault prediction research topic. The exclusion did not take into account the publication year of paper or methods which have been used.

2.2. Classification of papers

During our Fault Prediction Research Program, we noticed that software fault prediction papers can be categorized according to

* Corresponding author.
E-mail addresses: cagatay.catal@bte.mam.gov.tr (C. Catal), banu@ce.yildiz.edu.tr (B. Diri).

0957-4174/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2008.10.027

Table 1
Research questions.

Research question	Main motivation
RQ1: Which journal is the dominant software fault prediction journal?	Identify the most important software fault prediction journal
RQ2: What kind of datasets are the most used for fault prediction?	Identify whether predictive models are repeatable or not by examining the usage of public datasets
RQ3: What kind of datasets are the most used for fault prediction after year 2005?	Identify whether predictive models are repeatable or not by examining the usage of public datasets after year 2005
RQ4: What kind of methods are the most used for fault prediction?	Identify trends and opportunities for prediction method focus
RQ5: What kind of methods are the most used for fault prediction after year 2005?	Identify trends and opportunities for prediction method focus after year 2005
RQ6: What kind of metrics are the most used for fault prediction?	Identify trends and opportunities for prediction metrics focus
RQ7: What kind of metrics are the most used for fault prediction after year 2005?	Identify trends and opportunities for prediction metrics focus after year 2005
RQ8: What is the percentage of publications published after year 2000?	Identify if papers published after year 2000 represent the large portion of papers in literature or not

several criteria such as types of metrics, datasets and methods. For example, some researchers mostly used proprietary (private) datasets in their studies and this made it difficult to compare the performance of our models with the performance of models proposed in those papers. We classified the papers according to the properties and categories given in Table 2.

2.2.1. Metrics property

Metrics property has method-level, class-level, component-level, file-level, process-level, and quantitative-level categories. Method-level metrics are widely used for software fault prediction problem. Halstead (1977) and McCabe (1976) metrics have been proposed in 1970s but they are still the most popular method-level metrics. Method-level metrics can be collected for programs developed with structured programming or object-oriented programming paradigms because source code developed with these paradigms include methods. When method-level metrics are used, predicted fault-prone modules are the methods which likely to have faults during system testing or field tests. Widely used method-level metrics are shown in Table 3. PROMISE repository includes public datasets and these datasets have metrics shown in Table 3 for several NASA projects.

Class-level metrics are only used for object-oriented programs because class concept is the abstract characteristic of a thing (object) for only object-oriented paradigm. Chidamber–Kemerer (CK) metrics suite (Chidamber & Kemerer, 1994) proposed in 1994 is still the most popular class-level metrics suite being used by several software tool vendors and researchers working on fault prediction. In addition to CK metrics suite, there are more suites such as MOOD, QMOOD, and L&K. MOOD (metrics for object-oriented design) (Abreu & Carapuca, 1994) was proposed in 1994 and empirically validated in 1996 (Abreu & Melo, 1996). Lorenz and Kidd introduced eleven metrics (Lorenz & Kidd, 1994) and these metrics suite is known as L&K metrics suite. QMOOD (quality metrics for object-oriented design) was proposed by Bansiya and Davis (2002). However, CK metrics suite is much more popular than other suites and they are mostly used if class-level metrics are applied.

Table 2
Classification of papers.

Property	Categories
Year	Years between 1990 and 2007
Metrics	Method-level, class-level, component-level, file-level, process-level, quantitative values-level
Methods	Machine learning, statistics, machine learning + statistics, expert opinion
Evaluation criteria	Parameter from ROC curve, statistical parameters
Publication type	Journal article, proceeding, book chapter

Table 3
Widely used method-level metrics.

Attributes	Information
loc	McCabe's line count of code
v(g)	McCabe "cyclomatic complexity"
ev(g)	McCabe "essential complexity"
iv(g)	McCabe "design complexity"
n	Halstead total operators + operands
v	Halstead "volume"
l	Halstead "program length"
d	Halstead "difficulty"
i	Halstead "intelligence"
e	Halstead "effort"
b	Halstead "bug"
t	Halstead's time estimator
lOCode	Halstead's line count
lOComment	Halstead's count of lines of comments
lOBlank	Halstead's count of blank lines
lOCodeAndComment	Lines of comment and code
uniq_Op	Halstead Unique operators
uniq_Opnd	Halstead Unique operands
total_Op	Halstead Total operators
total_Opnd	Halstead Total operands
branchCount	Branch count of the flow graph

We used Artificial Immune Recognition System algorithm and CK metrics suite to predict fault-prone classes during our Fault Prediction Research Program (Catal & Diri, 2007b). Chidamber–Kemerer object-oriented metrics are Coupling_Between_Objects (CBO), Depth_Of_Inheritance_Tree (DIT), Lack_Of_Cohesion_Of_Methods (LCOM), Num_Of_Children (NOC), Response_For_Class (RFC), and Weighted_Method_Per_Class (WMC). WMC is the number of methods which locate in each class. DIT is the distance of the longest path from a class to the root in the inheritance tree. RFC is the number of methods which can be executed to respond a message. NOC is the number of classes which are direct descendants for each class. CBO is the number of non-inheritance-related classes to which a class is coupled. LCOM is related to the access ratio of attributes. According to several software fault prediction studies, CBO, WMC, and RFC are the most significant CK metrics for fault prediction (Zhou & Leung, 2006).

Method-level or class-level metrics can not be used for programs developed with component-based approach. Initial component-level metrics have been proposed in 2001 and researchers are still working on these kinds of metrics (Gill & Grover, 2003, 2004; Sedigh-Ali, Ghafoor, & Paul, 2001a, Sedigh-Ali, Ghafoor, & Paul, 2001b). Table 4 shows these component-level metrics (Mahmood et al., 2005).

Some researchers used metrics which have been collected per source file (Khoshgoftaar, Gao, & Szabo, 2001; Ostrand, Weyuker, & Bell, 2005) and we call these metrics "file-level metrics". Some

Table 4
Component-level metrics (Menzies et al., 2007).

Quality attribute	Metrics	Description
Complexity	Interface	An estimate of the complexity of interfaces
Complexity	Integration	Measure of effort required in the integration process of component-based system
Complexity	Semantic	Measure of complexity of the relationship of components to a given language
Portability	Adaptability	Opportunity to adoption to different environment
Portability	Replaceability	Opportunity and effort of using components in place of other components
Portability	Portability compliance	Adherence to application of portability related standard
Reliability	Maturity	Capacity to avoid failure as a result of faults in the component-based system
Reliability	Fault tolerance	Ability to maintain a specified level of performance in case of faults
Reliability	Recoverability	Capacity to re-establish level of performance after the faults
Functionality	Suitability	Presence of set of functions for specified tasks
Functionality	Accuracy	Provision of agreed results or effects
Functionality	Interoperability	Capability of the component-based system to interact with specified system
Maintainability	Analysability	Identification of deficiencies, failure causes, components to be modified
Maintainability	Changeability	Capability to enable a specified evolution to be implemented
Maintainability	Stability	Capability to avoid unexpected effects from evolution
Maintainability	Testability	Capability to enable validating the evolving component-based system

of these file-level metrics are number of times the source file was inspected prior to system test release, number of lines of code for the source file prior to coding phase (auto-generated code), number of lines of code for the source file prior to system test release, number of lines of commented code for the source prior to code (auto-generated), number of lines of commented code for the source file prior to system test release (Khoshgoftaar et al., 2001).

Method-level, class-level, component-level, and file-level metrics are all product metrics, but predictive models can be enhanced with process-level metrics. Jiang, Cukic, and Menzies (2007) combined requirement metrics with code metrics and reported that requirement metrics improve the performance of models that use code metrics. These metrics have been gathered from textural requirement specification documents by using ARM (automated requirement measurement) tool. ARM has been developed in NASA-Goddard Software Assurance Research Center. ARM parses requirement specification documents and parses each requirement with respect to several specification groups such as imperatives, continuances, directives, options, and weak phrase (Jiang et al., 2007). In addition to these requirement metrics, researchers may apply different process metrics such as programmer experience level, the number of defects found in reviews, the amount of time that the module spent in reviews, the number of test cases and unique test cases run that touched the module (Kaszycki, 1999).

It is possible to use some quantitative values such as cpu usage, and disk space used to predict software defects and we call these metrics "quantitative-level metrics". Bibi et al. (Bibi, Tsoumakas, Stamelos, & Vlahvas, 2006) used document quality, number of user, average transactions, cpu usage, and disk space used metrics for software defect prediction using regression via classification. Their datasets belong to a commercial bank in Finland and they concluded that regression via classification (RvC) is a robust technique for regression tasks that need explanations.

2.2.2. Methods property

Methods property has "machine learning", "statistics", "machine learning + statistical" approaches, and "statistics + expert opinion" categories. Machine learning algorithms are widely used for software fault prediction and some powerful algorithms for this problem are Näive Bayes (Menzies et al., 2007), Random Forests (Ma, Guo, & Cukic, 2006), and J48 (Koru & Liu, 2005). Some researchers apply statistical methods such as univariate or multivariate binary logistic regression to predict faults (Olague, Gholston, & Quattlebaum, 2007). However, statistical models are considered black-box solutions because the relationship between inputs and responses can not be seen easily (De Almeida & Matwin, 1999). Using statistical models which have been validated with a specific organization's data is not convenient to be used in another company because statistical models are very dependent on data (De Almeida & Matwin, 1999) and different organizations may have different organizational data features. It is very useful to apply statistical feature reduction techniques such as principal component analysis (PCA) to remove or reduce the multicollinearity among the metrics. Some researchers applied machine learning algorithms and statistical models during their researches and we classified these studies as "machine learning + statistical" approaches. The main motivation for fault prediction is to predict faults automatically, but some researchers also investigated the performance of expert estimations for this problem. It has been shown that statistical models outperform the expert estimations and experts can not work on very large datasets. However, experts can take into account the project issues.

2.3. Threats to validity

Both authors of this review paper mainly publish papers on software fault prediction based on machine learning models. As Jorgensen and Shepperd (2007) stated, we are not aware of biases we may have had for choosing the papers.

We did not search for papers based on issue-by-issue, manual reading of titles of all published papers in journals, as Jorgensen and Shepperd (2007) did. This means that we probably have excluded some fault prediction papers which locate in some journals or conference proceedings.

We did not exclude of conference papers because experience reports are mostly published in conference proceedings. Therefore, we could include a source of information about the industry's experience. Jorgensen and Shepperd (2007) did not use conference proceedings in their review because workload would increase significantly. They used 304 relevant journals on software development cost estimation, but history of software fault prediction research area is not old and not popular as cost estimation problem.

3. Results

Twenty-seven journal papers and 47 conference proceedings have been evaluated in this review systematically. Publication years of papers are between year 1990 and 2007. Fig. 1 is a curve which plots publication year on the *x*-axis and the number of papers published in that year on the *y*-axis for papers in review.

Sixty-one percentage of papers are conference proceedings, 36% of papers are journal papers, and 3% of papers are book chapters.

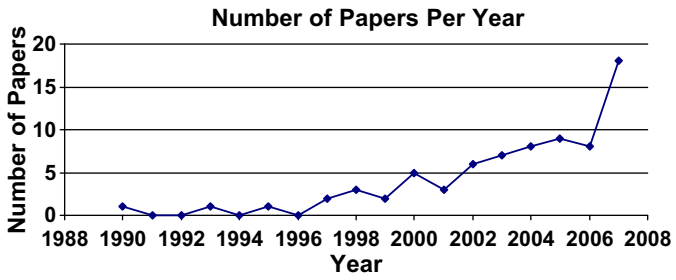

Fig. 1. Number of papers per year in review.

Each subsection of this section will address each research question given in Table 1.

3.1. Relevant fault prediction journals (RQ1)

We used papers on software fault prediction in as many as 11 journals and these journals with three or more papers on fault prediction are displayed in Table 5, together with the corresponding number, proportion, and cumulative proportions of papers.

Proportions and cumulative proportions have been calculated by considering only the number of journal papers in review. Four journals shown in Table 5 include 70% of all journal papers in review. Reading only the 4 most relevant journals means that important research results may be missed and this number is 10 in Jorgensen and Shepperd study (Jorgensen & Shepperd, 2007) for cost estimation.

3.2. Distribution of datasets (RQ2)

One of the most important problems for software fault prediction studies is the usage of non-public (private) datasets. Several companies developed fault prediction models using proprietary data and presented these models in conferences. However, it is not possible to compare results of such studies with results of our own models because their datasets can not be reached. Because machine learning researchers had similar problems in 1990s, they created a repository called UCI (University of California Irvine). Similar to UCI, Software Engineers have PROMISE repository which has several public datasets since 2005. NASA fault prediction datasets locate in PROMISE with a format (ARFF) that makes it possible to use these datasets directly from an open source machine learning tool called WEKA.

Papers have been divided into four categories regarding to their types of datasets: public, private, partial, and unknown. Public datasets mostly locate in PROMISE and NASA MDP (metrics data program) repositories and they are distributed freely. Private datasets belong to private companies and they are not distributed as public datasets. Partial datasets are datasets which have been created using data from open source projects and have not been distributed to community. These partial datasets do not locate in public repositories such as PROMISE. If there is no information about the dataset in the paper, the type of dataset is called "unknown". We identified four types of datasets for our review, but different researchers may use different groups. Fig. 2 shows the distribution of dataset types.

According to Fig. 2, 60% of papers have non-public (private) datasets. That's why; these papers are not repeatable, refutable, and verifiable. Unfortunately, 31% of papers have public datasets. This means that only one paper's result from three papers can be compared and it is repeatable. 23 public, 1 unknown, 44 private, and 6 partial datasets exist in papers.

Table 5
Most important software fault prediction journals.

Rank	Journal	Number	Proportion (%)	Cumulative proportion (%)
1	IEEE Transactions on Software Engineering	9	33	33
2	Software Quality Journal	4	15	48
3	Journal of Systems and Software	3	11	59
4	Empirical Software Engineering	3	11	70

3.3. Distribution of datasets after year 2005 (RQ3)

Distribution of datasets after year 2005 is shown in Fig. 3.

Because PROMISE repository has been created in 2005 and researchers started to be aware of public datasets, the usage of public datasets increased from 31% to 52%. Eighteen public, 1 unknown, 11 private, and 5 partial datasets exist in papers. This means that software engineers took one step further towards a mature software engineering discipline by using public datasets.

3.4. Distribution of methods (RQ4)

Distribution of methods which have been used in papers is shown in Fig. 4.

Methods have four groups: statistical methods, machine learning based methods, statistical methods + expert opinion, statistical methods + machine learning based methods. If machine learning based methods are used together with statistical methods in the same model or they are used separately in the paper, method of that paper is marked as "statistical methods + machine learning based methods". Similar to previous group, if statistical methods and expert opinion are used in the same paper, method of that paper is marked as "statistical methods + expert opinion". Fifty-

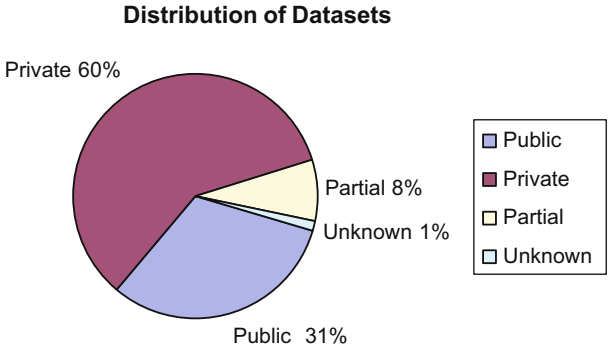

Fig. 2. Distribution of datasets.

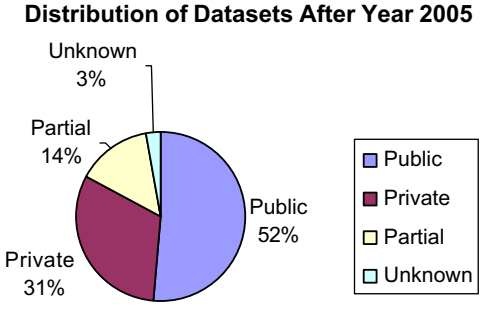

Fig. 3. Distribution of datasets after year 2005.

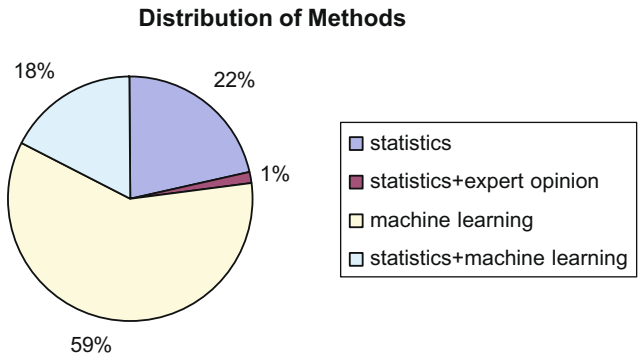

Fig. 4. Distribution of methods.

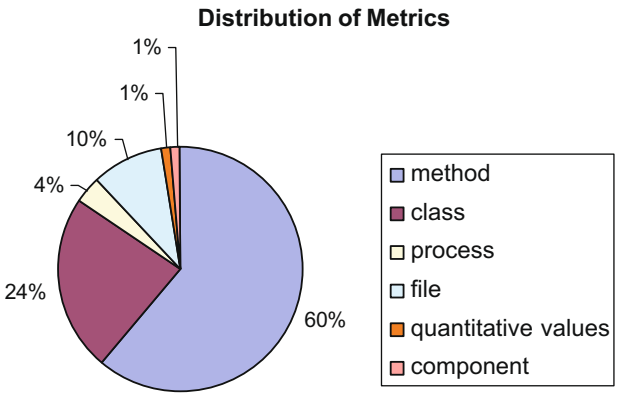

Fig. 6. Distribution of metrics.

nine percentage of papers used machine learning based methods and only 22% of papers used statistical methods as shown in Fig. 4. Because statistical methods are considered black-box solutions and these models are highly dependent on data, it is promising to see that more researchers are exploring the potential of machine learning methods to predict fault-prone modules. Expert opinion has been used in only one paper and its performance has been compared with statistical methods. Sixteen papers used statistical methods, 1 paper applied statistical methods and expert opinion, 44 papers used machine learning based methods, and 13 papers applied statistical methods together with machine learning based methods or separately.

3.5. Distribution of methods after year 2005 (RQ5)

Distribution of methods which have been used in papers after year 2005 is shown in Fig. 5.

According to the Fig. 5, it is easily seen that the percentage of papers which used machine learning based methods increased from 59% to 66%. The percentage of papers which used statistical methods decreased from 18% to 14%. This means that researchers started to apply more machine learning based models and they have less motivation to use statistical methods regarding their deficiencies in the area of software fault prediction. Five papers used statistical methods, 1 paper applied statistical methods and expert opinion, 23 papers used machine learning based methods, and 5 papers applied machine learning based methods together with statistical methods or separately.

3.6. Distribution of metrics (RQ6)

Distribution of metrics which have been used in papers is shown in Fig. 6. Metrics have six groups: method-level, class-level, process-level, component-level, file-level, and quantitative-level metrics.

According to the Fig. 6, 60% of papers used method-level metrics and 24% of papers applied class-level metrics. Only 4% of papers have process-level metrics. When method-level metrics are used instead of class-level metrics, predicted fault-prone modules are methods instead of classes. Therefore, more specific parts of source code (methods) which are fault-prone can be identified. Furthermore, method-level metrics are widely used since 1970s, but class-level metrics became popular in the late 1990s. So this may explain why the usage of method-level metrics is higher than other level metrics. Fifty-one papers had method-level metrics and 20 papers had class-level metrics. Three papers used process-level metrics, 8 papers applied file-level metrics, 1 paper used component-level, and 1 paper applied quantitative-level metrics.

3.7. Distribution of metrics after year 2005 (RQ7)

Distribution of metrics which have been used in papers after year 2005 is shown in Fig. 7.

According to the Fig. 7, the usage percentage of class-level metrics is same. However, the usage percentage of method-level metrics increased, and the usage percentage of file-level metrics increased. The percentages of papers having process or component-level metrics are still very low. Twenty-two papers used method-level metrics and 10 papers used class-level metrics. 1 paper had process-level metrics, 7 papers had file-level metrics, and 1 paper used quantitative-level metrics.

3.8. Distribution of papers after year 2000 (RQ8)

We examined papers regarding to their publication date. Papers have been classified into two groups: papers published before year

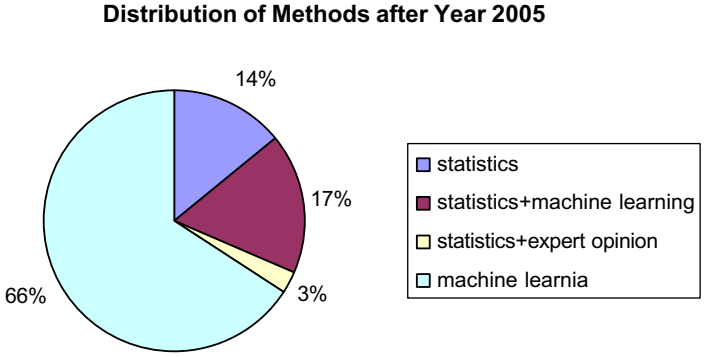

Fig. 5. Distribution of methods after year 2005.

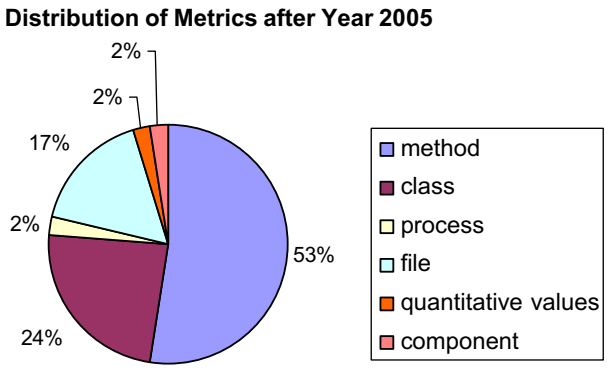

Fig. 7. Distribution of metrics after year 2005.

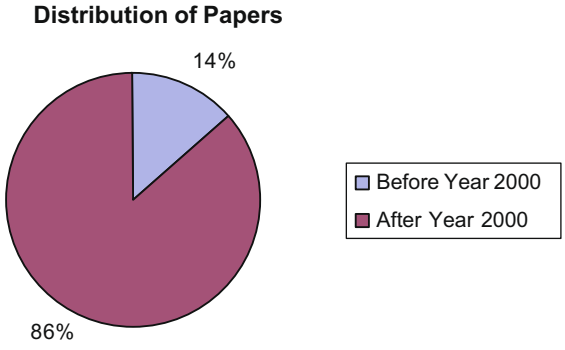

Fig. 8. Distribution of papers.

2000 and papers published after year 2000. Fig. 8 shows the distribution of papers regarding to the publication date.

Ten papers have been published before year 2000 and 64 papers have been published after year 2000. It is clearly seen that the popularity of fault prediction method increased drastically after year 200 and we can advocate that researchers should only examine papers published after year 2000 to reach the most important papers.

4. Conclusion

This paper reviewed software fault prediction papers published in conference proceedings and journals to evaluate the progress and direct future research on software fault prediction. We evaluated papers with a specific focus on types of metrics, methods, and datasets and did not describe all the prediction models in detail. The aim was to classify studies with respect to metrics, methods, and datasets that have been used in fault prediction papers. We evaluated papers published before and after 2005 with respect to metrics, methods, and datasets.

We suggest the following changes in software fault prediction research:

- *Conduct more studies on fault prediction models using class-level metrics.* Even though object-oriented paradigm is widely used in industry, the usage percentage of class-level metrics are still beyond acceptable levels. We need prediction models using class-level metrics to predict faults during design phase and this type of prediction is called early prediction. In addition to class-level metrics, the usage percentages of component-level and process-level metrics are very low. It is an open research area to investigate component-level or process-level metrics for fault prediction problem.
- *Increase the usage of public datasets for fault prediction problem.* It is very significant to use public datasets for fault prediction because repeatable, refutable and verifiable models of software engineering can only be built with public datasets. However, the usage percentage of public datasets is 31% for this review and it is 52% for papers published after year 2005. Therefore, we need to increase the percentage of papers using public datasets and 80% can be an ideal level.
- *Increase the models based on machine learning techniques.* As specified in this review, machine learning models have better features than statistical methods or expert opinion based approaches. Therefore, we should increase the percentage usage of the models based on machine learning techniques.

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.

Appendix A. List of included papers

(The numbers of papers are shown in ( ). The titles of journals with more than two papers appear in italic.)

Empirical Software Engineering (3) – Rank 4
IEEE Software (2)
IEEE Transactions for Reliability (1)
IEEE Transactions on Neural Networks (1)
IEEE Transactions on Software Engineering (9) – Rank 1
IEEE Transactions on Systems, Man, Cybernetics (1)
International Journal of Software Engineering and Knowledge Engineering (1)
Journal of Systems and Software (3) – Rank 3
SIGSOFT Software Engineering Notes (1)
Software Quality Journal (4) – Rank 2
Software-Practice and Experience (1)

Appendix B. Papers on Software Fault Prediction in Review

1. Porter, A. A., & Selby, R. W. (1990). Empirically guided software development using metric-based classification trees. *IEEE Software, 7*(2), 46–54.
2. 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.
3. 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).
4. 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).
5. 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.
6. Evett, M., Khoshgoftar, T., Chien, P., & Allen, E. GP-based software quality prediction. In *Third annual conference on genetic programming* (pp. 60–65).

7. Ohlsson, N., Zhao, M., & Helander, M. (1998). Application of multivariate analysis for software fault prediction. *Software Quality Journal, 7*(1), 51–66.
8. Binkley, A. B., & Schach, S. R. (1998). Prediction of run-time failures using static product quality metrics. *Software Quality Journal, 7*(2), 141–147.
9. 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).
10. Kaszycki, G. (1999). Using process metrics to enhance software fault prediction models. In *Tenth international symposium on software reliability engineering, Boca Raton, Florida*.
11. 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* (p. 85). Washington, DC: IEEE Computer Society.
12. 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.
13. 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.
14. 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).
15. 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.
16. 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.
17. Schneidewind, N. F. (2001). Investigation of logistic regression as a discriminant of software quality. In *Seventh international symposium on software metrics* (328–337). Washington, DC: IEEE Computer Society.
18. 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.
19. 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.
20. 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.
21. 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).
22. Khoshgoftaar, T. M., & Seliya, N. (2002). 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.
23. 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.
24. Khoshgoftaar, T. M., & Seliya, N. (2002). Tree-based software quality estimation models for fault prediction. In *Eighth IEEE symposium on software metrics, Ottawa, Canada* (pp. 203–215).
25. 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).
26. 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.
27. 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*.
28. 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, Amsterdam, The Netherlands* (pp. 113–122). IEEE Computer Society.
29. Khoshgoftaar, T. M., & Seliya, N. (2003). Fault prediction modeling for software quality estimation: Comparing commonly used techniques. *Empirical Software Engineering, 8*(3), 255–283.
30. 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.
31. 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.
32. Menzies, T., DiStefano, J., Orrego, A., & Chapman, R. (2004). Assessing predictors of software defects. In *Predictive software models workshop*.
33. Khoshgoftaar, T. M., & Seliya, N. (2004). Comparative assessment of software quality classification techniques: An empirical case study. *Empirical Software Engineering, 9*(3), 229–257.
34. 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.
35. 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).
36. 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.
37. 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*.
38. 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.

39. 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).
40. 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.
41. Koru, A. G., & Liu, H. (2005). An investigation of the effect of module size on defect prediction using static measures. In *Workshop on predictor models in software engineering, St. Louis, Missouri* (pp. 1–5).
42. 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.
43. Koru, G., & Liu, H. (2005). Building effective defect prediction models in practice IEEE software. *IEEE Software, 22*(6), 23–29.
44. 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.
45. 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.
46. 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.
47. 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).
48. 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.
49. Ma, Y., Guo, L., & Cukic, B. (2006). A statistical framework for the prediction of fault-proneness. *Advances in machine learning application in software engineering* (pp. 237–265). Idea Group Inc.
50. 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).
51. 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.
52. Nikora, A. P., & Munson, J. C. (2006). Building high-quality software fault predictors. *Software-Practice and Experience, 36*(9), 949–969.
53. 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.
54. Mertik, M., Lenic, M., Stiglic, G., & Kokol, P. (2006). Estimating software quality with advanced data mining techniques. In *International conference on software engineering advances* (p. 19). Papeete, Tahiti, French Polynesia: IEEE Computer Society.
55. 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.
56. 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.
57. 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.
58. 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).
59. Menzies, T., Dekhtyar, A., Distefano, J., & Greenwald, J. (2007). 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.
60. 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).
61. 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).
62. 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.
63. Menzies, T., Greenwald, J., & Frank, A. (2007). Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering, 33*(1), 2–13.
64. Yang, B., Yao, L., & Huang, H. Z. Early software quality prediction based on a fuzzy neural network model. In *Third international conference on natural computation, Haikou, Çin* (pp. 760–764).
65. 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.
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.
67. 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.
68. 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.
69. Cukic, B., & Ma, Y., (2007). Predicting fault-proneness: Do we finally know how? In *Reliability analysis of system failure data, Cambridge, UK*.
70. Binkley, D., Feild, H., & Lawrie, D. (2007). Software fault prediction using language processing. Testing: Industrial and academic conference. *Practice and research techniques* (pp. 99–108). Cumberland Lodge, Windsor, UK: IEEE Press.
71. Seliya, N., & Khoshgoftaar T. M. (2007). Software quality analysis of unlabeled program modules with semisupervised clustering. *IEEE Transactions on Systems, Man, and Cybernetics, 37*(2), 201–211.
72. Seliya, N., & Khoshgoftaar T. M. (2007). Software quality estimation with limited fault data: A semi-supervised learning perspective. *Software Quality Journal, 15*(3), 327–344.
73. 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.

74. 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*.

References

Abreu, F. B. e., & Carapuca, R. (1994). Object-oriented software engineering: Measuring and controlling the development process. In *Fourth international conference on software quality, McLean, VA, USA*.

Abreu, F. B. e., & Melo, W. (1996). Evaluating the impact of object-oriented design on software quality. In *Proceedings of the 3rd international symposium on software metrics: From measurement to empirical results, Berlin, Germany* (p. 90).

Bansiya, J., & Davis, C. G. (2002). A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering, 28*(1), 4–17.

Bibi, S., Tsoumakas, G., Stamelos, I., & Vlahvas, I. (2006). Software defect prediction using regression via classification. In *IEEE international conference on computer systems and applications* (pp. 330–336). Dubai, UAE: IEEE Computer Society.

Catal, C., & Diri, B. (2007a). Software defect prediction using artificial immune recognition system. In *Proceedings of the IASTED international conference on software engineering, IASTED, Innsbruck, Austria* (pp. 285–290).

Catal, C., & Diri, B. (2007b). Software fault prediction with object-oriented metrics based artificial immune recognition system. In *Eighth international conference on product focused software process improvement. Lecture notes in computer science* (pp. 300–314). Riga, Latvia: Springer-Verlag.

Catal, C., & Diri, B. (2008). A fault prediction model with limited fault data to improve test process. Product focused software process improvement. In *Proceedings of the 9th international conference on product focused software process improvement. Lecture notes in computer science*. Rome, Italy: Springer-Verlag.

Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object-oriented design. *IEEE Transactions on Software Engineering, 20*(6), 476–493.

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

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.

El Emam, K., Benlarbi, S., Goel, N., & Rai, S. (2001). Comparing case-based reasoning classifiers for predicting high risk software components. *Journal of Systems and Software, 55*(3), 301–320.

Evett, M., Khoshgoftaar, T., Chien, P., & Allen, E. (1998). GP-based software quality prediction. In *Proceedings of the 3rd annual genetic programming conference, San Francisco, CA* (pp. 60–65).

Gill, N. S., & Grover, P. S. (2003). Component based measurement: Few useful guidelines. *SIGSOFT Software Engineering Notes, 28*(6), 1–6.

Gill, N. S., & Grover, P. S. (2004). Few important considerations for deriving interface complexity metric for component-based systems. *SIGSOFT Software Engineering Notes, 29*(2), 4.

Guo, L., Cukic, B., & Singh, H. (2003). Predicting fault prone modules by the Dempster–Shafer belief networks. In *Proceedings of the 18th IEEE international conference on automated software engineering* (pp. 249–252). Montreal, Canada: IEEE Computer Society.

Halstead, M. (1977). *Elements of software science*. New York: Elsevier North-Holland.

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.

Jorgensen, M., & Shepperd, M. (2007). A systematic review of software development cost estimation studies. *IEEE Transactions on Software Engineering, 33*, 33–53.

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., 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.

Khoshgoftaar, T. M., & Seliya, N. (2002). Software quality classification modeling using the SPRINT decision tree algorithm. In *Proceedings of the 4th IEEE international conference on tools with artificial intelligence, Washington, DC* (pp. 365–374).

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

Lorenz, M., & Kidd, J. (1994). *Object-oriented software metrics: A practical guide*. Prentice-Hall, Inc.

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).

Mahmood, S., Lai, R., Kim, Y. S., Kim, J. H., Park, S. C., & Oh, H. S. (2005). A survey of component based system quality assurance and assessment. *Information and Software Technology, 47*(10), 693–707.

McCabe, T. (1976). A complexity measure. *IEEE Transactions on Software Engineering, 2*(4), 308–320.

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

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.

Sayyad, S. J., & Menzies, T. J. (2005). *The PROMISE repository of software engineering databases*. Canada: University of Ottawa. <http://promise.site.uottawa.ca/SERepository>.

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.

Sedigh-Ali, S., Ghafoor, A., & Paul, R. A. (2001a). Metrics-guided quality management for component-based software systems. In *Proceedings of 25th annual international on computer software and applications conference* (pp. 303–308).

Sedigh-Ali, S., Ghafoor, A., & Paul, R. A. (2001b). Software engineering metrics for COTS-based systems. *IEEE Computer, 34*(5), 44–50.

Thwin, M. M., & Quah, T. (2003). Application of neural networks for software quality prediction using object-oriented metrics. In *Proceedings of the 19th international conference on software maintenance, Amsterdam, The Netherlands* (pp. 113–122).

Yuan, X., Khoshgoftaar, T. M., Allen, E. B., & Ganesan, K. (2000). An application of fuzzy clustering to software quality prediction. In *Proceedings of the 3rd IEEE symposium on application-specific systems and software engineering technology* (pp. 85). Washington, DC: IEEE Computer Society.

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.