Lessons from applying experimentation in software engineering prediction systems

Wasif Afzal, Richard Torkar
Blekinge Institute of Technology,
S-372 25 Ronneby, Sweden
{waf,rto}@bth.se

Abstract

Within software engineering prediction systems, experiments are undertaken primarily to investigate relationships and to measure/compare models' accuracy. This paper discusses our experience and presents useful lessons/guidelines in experimenting with software engineering prediction systems. For this purpose, we use a typical software engineering experimentation process as a baseline. We found that the typical experimentation process in software engineering is supportive in developing prediction systems and have highlighted issues more central to the domain of software engineering prediction systems.

1 Introduction

Software development process is costly and therefore demands efficient allocation of resources. Measurement during the different phases of a software development process makes different activities visible and hence provides opportunities for making efficiency gains, not necessarily limited to resource allocation. There has been significant interest in the software engineering research community to make use of these measures for predicting the future outcomes, in the form of prediction systems and models. Models act as a substitute for the complex real-world systems, to help us better represent the realities [12]. Statistical models are more common to software engineering whereby they are utilized for prediction of probabilistic future behavior of a system using prior data and extrapolation or interpolation of data based on a mathematical fit. There have been many published results on software effort/cost estimation and software fault/fault count predictions. With so many studies, the problem of reliable predictions is still largely unsolvable as we are not able to reach a general conclusion due to contradicting empirical results. This shows that there is enough uncertainty in the software prediction process that hampers reaching consistent and reliable results.One of the obvious reasons for having such a variation in the prediction results is that the software development process is seldom repeatable. Each new software project is innovative and many times solves new problems. Therefore, we have to deal with complex relationships among several variables and the interaction of application with its environment [14]. However, this is not the only problem facing inaccurate predictions in software engineering.

This study, in addition to summarizing some of the key reasons for inaccurate and contradictory prediction results, presents lessons/guidelines based on our experience in experimenting with prediction systems and related literature investigation. We use the typical experimentation process (given in [35]) as a baseline to the process of building software engineering prediction systems and to present critical issues specific to software prediction studies. The motivation for using a process-centric approach is that the research procedure quality is bound to impact the conclusions of a study. Myrtveit et al. [26] argues that lack of convergence of studies on software prediction models is partly attributed to variation in their quality of execution. Therefore, by improving the experimentation process we can achieve better convergence of results and improve validity of conclusions.

2 Experimental software engineering

Experiments in software engineering is part of a wider context i.e. empiricism in software engineering. The important reasons for undertaking quantitative empirical studies (i.e. experiments and case studies) are summarized by Wohlin et al. [35] as, "to get objective and statistically significant results regarding the understanding, controlling, prediction and improvement of software development".

In software engineering, the steps required to perform experiments have been documented in a dedicated book to help software engineers in performing experiments [35]. According to this book, the steps constituting the process

of experimentation include: definition; planning; operation; analysis and interpretation; presentation and package.

The representation of the experimental process in above steps manifest that experimentation is a formal and controlled activity [35]. There are few studies that report experimentation in software engineering, Sjøberg et al. [31] found 103 papers out of 5453, taken from 12 conferences and journals, that could be categorized as being experiments [18]. Similar results are also reported by [33].

Following is a summary of different experimental steps [35]:

Definition: The definition step helps defining the goals and objectives of the experiment. This is one of the foundation steps for experimentation. *Planning:* The planning step includes determination of experiment context, formal statement of hypothesis, selection of variables and subjects, selecting experimental design, instrumentation and validity evaluation. *Operation:* The experiment operation consists of preparation, execution and data validation. *Analysis and interpretation:* The first step in analysis is to use descriptive statistics to provide a visualization of data. The second step is data reduction and the third step is hypothesis testing. *Presentation and package:* This step concerns the documentation of the experimental process and final results.

We use this experimentation process as a baseline to present the association with prediction studies in software engineering.

3 Software engineering prediction systems

There are many opportunities of making use of prediction systems in software engineering. Fenton and Pfleeger [11] show that predictions are needed throughout the software development life cycle, from feasibility through maintenance. The two most commonly targeted areas of prediction in software engineering are project effort and faults [29].

Software engineering literature has established several benefits out of accurate predictions and estimates:

1. Accurate cost estimation eliminates chances of overrun budgets and schedules. Similarly, overestimation can be avoided to achieve time and resource efficiencies [16].

2. Timely identification of fault-prone modules assists in an efficient allocation of testing resources [20] and prioritization of efforts [17].

3. The identification of fault-prone modules may trigger more thorough design of risky components, thus improving software architecture [20].

4. The prediction of fault count data helps predicting the quality to be achieved from a software [10].

5. "A good defect prediction model is an important first step towards pricing maintenance contracts, estimating support costs such as maintenance staffing, and creating software insurance" [21].

6. Software reliability prediction in terms of prediction of faults is indispensable with respect to determining optimal time to stop testing [32].

In short, accurate predictions are helpful for "...tendering bids, monitoring progress, scheduling resources and evaluating risk factors" [21].

Keeping in view the above benefits, there are several predictive models proposed in software engineering literature. These models range from traditional statistical (regression) models to machine learning models and models making use of both traditional and machine learning techniques. Despite the presence of these many models, there has not been consensus in the research community regarding which approach is the most suitable one. "Indeed, very contradictory results have been reported in studies comparing an arbitrary function approximator (or machine learning model) with a function. Furthermore, the performance of arbitrary function approximators varies widly across studies" [26].

The most recent study that we know of, related to fault predictions, is by Lessmann et al. [20], who established that metric-based classification is useful. But still the authors suggest more research to improve convergence across studies.

There has been considerable interest in understanding the reasons behind contradicting results in software engineering prediction studies. Several studies have identified various reasons attributed to diverging results of software engineering prediction studies [26, 20, 16, 9]. A brief summary of such reasons is given below.

Nature of software engineering data sets. Software engineering data sets have properties that challenge effective analysis and modeling. These properties include missing data, presence of many explanatory variables (both continuous and discrete), complex interdependencies and collinearity between the variables, heteroscedasticity, presence of outliers (or atypical variable values) and small size of data [4, 29, 30]. Although some of these issues can be reduced to some extent (discussed later in Section 4), others cannot be and thus any model derived using such data would be less reliable.

Incomplete understanding of software development process. Since software data is complex and have many variables, we possess a poor understanding of the software development process. Therefore, "...it is very difficult to make valid assumptions about the form of the functional relationship between the variables" [4]. Since variables selection is an important part of experiment planning, any short-

coming at this step is expected to bias the experimental results.

Misleading accuracy indicators. The software engineering research community has realized that there are certain accuracy indicators that are not only invalid but also unreliable. There are studies indicating that there is no consensus among use of various accuracy indicators and the choice of indicator determines the preferred prediction system which is of course not desirable [19, 25, 29].

4 Lessons learned and guidelines

In this section, we summarize our experiences and present useful lessons with respect to the different steps in the typical software engineering experimental process (Section 2) and the predictive modeling studies.

4.1 Definition

This step defines the basic purpose of the experiment and guides rest of the experimentation process. We found the use of the goal definition template [35] as a useful first step in experimenting with prediction models because it takes into account important aspects of objects, purpose, quality focus and perspective. We summarize one of our studies [3] in the goal definition template:

"Analyze the *traditional and genetic programming (GP) techniques* [objects]
for the purpose of *evaluation* [purpose]
with respect to *model validity, goodness of fit and model bias* [quality focus]
from the point of view of *the researcher* [perspective]
in the context of *fault count data from three industrial projects* [context]"

We found the definition step to be as important in experimenting with predictive models as generally in software engineering experimentation.

4.1.1 Useful lessons

1. The objects define the scope of the experiment [35], therefore a background in related literature helps to clarify the need of an experiment, such as the expected contribution of the experiment in increasing our understanding of the trade-offs among different prediction systems.

2. The definition step is expected to be refined and revised before experiment operation due to a gradual increase in problem understanding during planning.

4.2 Planning

Once the need of an experiment is identified, it is important to properly plan the experiment. Context selection [35] is one of the steps in experiment planning. With respect to context, we were engaged in performing experiments in an on-line situation, which in our case, meant that the experiments were based on the data collected from real-world projects. On the other hand, other context are also possible, like making use of replication and simulation.

Simulation can be a possible way out of the problem of having limited data. Pickard et al. [28] were the first to propose the use of simulation for evaluating software models [30]. Simulation can be used in different situations, i.e. a large number of values can be created that follow a particular data distribution e.g. Gaussian. It is also lot easier to compare a true distribution of data against another one and it enables study of complex phenomenon where analytical solutions are difficult to reach [13]. Shepperd et al. [29] also recommend simulation, as a way to counter difficulties when collecting large industrial data sets.

We also consider *replication* of studies as an important research methodology to better understand and generalize the study results. A replicated study helps to identify any anomaly or similarity among study results and an insight into the factors causing it. Especially with contradicting results in prediction studies in software engineering, replication would contribute to possibly present different perspectives on the problem for an increased understanding.

Within selecting context for an experiment, it is often required to select models or methods for comparison. Within prediction studies in software engineering, it is important to select a representative set of models for comparisons. Having a representative set of comparative models would increase generalizability of results and conclusion validity. There can be several motivations of selecting comparative models which are largely derived by the research hypothesis. As an example, in one of our studies [3], three fault count models were selected for comparison as the goal was to compare a family of fault count models. Also, these models presented a fair representation in terms of different forms of the growth curve.

After context selection, a hypothesis is formally stated which is later validated using statistical tests. We formulated the following null and alternative hypotheses in [3]:

$H_{0-\mathrm{gof}}$: The GP evolved model does not give significantly higher goodness of fit as compared with traditional models.

$H_{1-\mathrm{gof}}$: The GP evolved model gives significantly higher goodness of fit as compared with traditional models.

We also need to select the independent and dependent variables. Independent variables are those that are manipulated and controlled; and their effect is measured in depen-

dent variables [35]. This is an important step in experimental planning, especially in metric-based regression models.

The selection of subjects [35] is also an important step in experimental planning. In prediction studies, subjects might vary; e.g., in one of our studies [3], we used weekly fault count data sets collected during the testing of three large scale software projects as the subjects. Since selection of subjects is related to the ability to generalize the results [35], we are not certain in stating what is a reasonable level of generalizability. We plan to investigate this aspect further in future studies.

The planning of an experiment also include selecting an experimental design that is suitable for using statistical analysis [35]. We used one factor with two treatments in one of our studies [3] which compared fault prediction using GP with three traditional fault prediction models. So in this case, factor is the fault prediction method and the treatments are use of GP and traditional models. As part of design, the hypothesis should be analyzed to select appropriate statistical analysis method [35]. In the formulated hypotheses in our study ($\mathrm{H}_{0-\mathrm{gof}}$, $\mathrm{H}_{1-\mathrm{gof}}$), it is evident that we need to use a goodness of fit test for testing the stated hypotheses.

It is also important to select instrumentation (objects, guidelines and measurement instruments) [35] for the experiment. The object in the case of prediction studies in software engineering might be fault data sets, while there needs to be some preparation (guidelines) if new methods are to be experimented. If there is a need for data collection, measurement instruments need to be developed, e.g., use of forms [35]. In building prediction systems, data collection might be in the form of model outputs on validation portion of the data set.

The last step with experiment planning is validity evaluation which is discussed in much detail in [35] along with the validity threats for conclusion, internal, construct and external validity. We realized that these validity threats are equally applicable to prediction studies in software engineering as in experimental software engineering in general.

4.2.1 Useful lessons

1. Data sets required for experimentation are difficult to get due to several reasons, e.g., data being confidential and lack of enough information to be extracted from the data. Simulation can be used to generate data that follows a particular distribution and may help reaching valid conclusions [28, 13, 29]. It is also possible to get publicly available data sets, e.g., PROMISE data sets [2] and from NASA IV&V Tools Lab [1].

2. Some considerations in selecting models for comparison are helpful:
   (a) Availability of software implementing the model algorithms.
   (b) Active research in a particular modeling mechanism (as in [30]).
   (c) Specific data requirements of a particular model, e.g. Shooman's exponential model's hazard function requires knowing the parameters of total number of instructions in the program and debugging time since the start of system integration.

3. The stated hypotheses need to be specific so that it can either be refuted or accepted using statistical tests.

4. It is also important to understand the process variables and their alignment with the context in focus so as to identify the correct experimental factors.

5. The decisions regarding the selection of variables, their scale type, stated hypotheses and types of statistical tests to perform are not independent but inter-related.

4.3 Operation

In the operational step, the subjects are exposed to treatments [35]. The operation step consists of three further steps of preparation, execution and data validation [35]. In the context of experimental software engineering in general, the preparation step would mean dealing effectively with *human* subjects on most of the occasions. On the other hand, it is not necessary in prediction studies in software engineering to involve human subjects. Thus the preparation step here would resemble the instrumentation step in the planning phase of experimental software engineering which includes choosing objects, guidelines and measurement instruments.

As part of the preparation steps, especially in prediction studies in software engineering, it might be a reasonable approach to do data preprocessing. Within machine learning techniques, data preprocessing may lead to improved results. The preprocessing of data using attribute selection, attribute discretization, data transformation and data cleansing [34] might increase chances of improved results. Attribute selection works to eliminate irrelevant attributes and can be done both manually and automatically. Attribute discretization is a kind of data transformation that involves converting numeric attributes into small number of distinct ranges [34] to make them suitable for some classification algorithms and finally, data cleansing refers to various ways to make data noise-free. Therefore, it seems useful to analyze the data sets to avail opportunities to make the data more suitable for different techniques. For example, factor analysis can be used to remove multicolinearity which causes incorrect statistical tests and misleading coefficient signs.

In the execution step of experiment operation, a typical software engineering experiment would collect experimental data to be used for statistical analysis. But as mentioned earlier, the execution step of experiments in predictive models mostly deals with model outputs on validation part of the data set. This also serves the purpose of data validation which is the third step in the experimental operation [35].

It is common in research on prediction models to divide the data set into training (or fix) and test set. To increase expectancy of unbiased results, an impartial data splitting or cross-validation technique is desirable. There are typically two types of cross-validation, n-fold (leave-one-out) and v-fold [26], where n is the number of instances in the data set and v is some number smaller then n. In n-fold cross-validation, the learning method is trained on all, except one, instances. The model's correctness is then evaluated on the remaining instance. In this way the results for all n members of the data set are averaged to present a final error estimate [34]. In v-fold cross-validation, the data set is split in to v partitions and each partition in turn is used for testing and the remainder is used for training. Standard practice is to use $v = 10$ for a 10-fold cross-validation. There is still no consensus on which cross-validation method is the most suitable. Myrtveit et al. [26] describes n-fold as being more practically suited to real-world software development situations than v-fold, but on the other hand, v-fold is less computationally intensive. However, any form of cross-validation will increase the transparency and unbiasedness of model results.

4.3.1 Useful lessons

1. Preprocessing of data might help a machine learning algorithm in converging to a suitable solution.
2. It is important to keep the training and test sets as independent because the validation of the learned model on an independent test set is expected to closely match the fresh data that will be applied in practice [34].
3. Within machine learning and evolutionary computation approaches to prediction studies, it is expected that some experimentation at the operational step might be required for adjusting the algorithmic parameters.
4. For machine learning approaches to predictive models, it is especially important to document the algorithmic settings and control parameters during the operation so as to serve as a basis for reaching optimal tuning of these parameters in future comparative studies.

4.4 Analysis and interpretation

Once data is collected during experiment operation, analysis and interpretation step can begin. Analysis and interpretation is done in three steps: descriptive statistics, reducing the data set and hypothesis testing [35].

Descriptive statistics are used for knowing the data distribution. Graphical visualization in the form of scatter plots, box plots and histograms illustrates the properties of data sets [35]. Therefore, descriptive statistics represent useful tools for data exploration.

Data set reduction deals with detecting outliers; while hypothesis testing makes use of parametric and non-parametric methods to test the formulated hypotheses.

Earlier studies on predictive accuracy of competing models did not use to test results for statistical significance and drew conclusions without reporting significance levels. This is, however, now less practiced as more and more studies report statistical tests of significance, e.g. in [15] one-way ANOVA and Tuckey's multiple comparison tests were used to analyze the predictive performances of the different methods with respect to the absolute average error (AAE) and absolute relative error (ARE) values. Statistical tests of significance are important since it is not reliable to draw conclusions merely on observed differences in means or medians because the differences could have been caused by chance alone [25]. The use of statistical tests of significance comes with its own share of challenges about which tests are suitable for a given problem. A study by Demšar [8] recommends non-parametric tests for statistical comparisons of classifiers; while elsewhere in [5] parametric techniques are seen as robust to limited violations in assumptions and as more powerful (in terms of sensitivity to detect significant outcomes) than non-parametric.

As we discussed earlier in Section 3, there is no consensus with regards as to which accuracy indicator is the most suitable for the problem at hand. Commonly used indicators suffer from different limitations, for details see [13, 29]. One intuitive way out of this dilemma is to employ more than one accuracy indicator, so as to better reflect on a model's predictive performance in light of different limitations of each accuracy indicator. This way the results can be better assessed with respect to each accuracy indicator and we can better reflect on a particular model's reliability and validity. However, reporting several measures that are all based on a basic measure like mean relative error (MRE) would not be useful [13]. In [27], measures for the following characteristics are proposed: Goodness of fit (Kolmogorov-Smirnov test), Model bias (U-plot), Model bias trend (Y-plot) and Short-term predictability (Prequential likelihood). These measures, although providing a thorough evaluation of a model's predictions, lacks a suitable measure for variable-term predictability. In [15, 24], average relative error is used as a measure of variable term predictability. To our knowledge, we are not aware of any critique of such an approach for variable-term predictability.

As an example of applying multiple measures, one of our recent studies [3] used measures of prequential likelihood, Braun statistic and adjusted mean square error for evaluating model validity. Additionally we examined the distribution to residuals from each model to measure model bias. Lastly, the Kolmogorov-Smirnov test was applied for evaluating goodness of fit. More recently, analyzing distribution of residuals is proposed as an alternative measure [19, 29]. It has the convenience of applying significance tests and visualizing differences in absolute residuals of competing models using box plots.

We see examples of studies in which the authors use a two-prong evaluation strategy for comparing various modeling techniques. They include both quantitative evaluation and subjective qualitative criteria based evaluation because they consider using only empirical evaluation as an insufficient way to judge a model's output accuracy. Qualitative criterion-based evaluation evaluates each method based on conceptual requirements [16]. One or more of these requirements might influence model selection. Examples of qualitative criteria include [6, 16, 22, 23]:

1. Does the model require specification of the form of relationship between the variables or does it determine its own structure?
2. How robust is the model in dealing with outliers (insensitivity to noise)?
3. Is the model's output affected by small data sets, and if yes, how much?
4. Can the model adjust to incorporate additional data or does the model require regeneration on the combined data set?
5. Is the process of model building transparent and reasoning process visible?
6. Is the model able to capture complex relationships in data?
7. Is the model capable of including known facts to improve and refine its output?
8. How easy is it to configure the technique used for modeling (ease of configuration)?
9. What time and memory resources are required for model building?
10. What is the extent of generality of model results for diverse data sets?
11. What is the applicability of the model in different lifecycle phases?

We can assign subjective ratings to above mentioned criteria to better explain the trade-offs in selecting a particular model. We argue that both quantitative and qualitative factors play an important part to establish the validity of a model. Therefore it should be possible to give a definite structure to these concepts to develop a multidimensional model evaluation system which gives proportionate weighing to empirical and qualitative factors for model assessment.

4.4.1 Useful lessons

1. It is important to satisfy the assumptions of statistical tests before considering to apply them in practice. According to Fenton et al. [9], one of the main reasons of invalid conclusions in empirical studies is not satisfying the assumptions of a statistical technique. One of the important assumptions of using appropriate statistics is the scale type of measure. As pointed out in [5], it is not always easy to figure out the scale type of measures in software engineering. This complicates the decision of applying parametric vs. non-parametric tests; but Briand et al. [5] demonstrates that if a researcher is confident that the scale type is between ordinal and interval, then it could be treated as being on an interval scale because commonly used parametric tests are robust to *non-linear* (not exponential) transformations of the interval scale.

2. Another decision to take is to select a suitable significance level (α) for hypothesis testing, though the most commonly used significance levels in software engineering are 0.01 and 0.05. The significance level shows the probability of committing a Type I error (incorrect rejection of null hypothesis). The test is more sensitive if lower values of α are chosen and chances of finding significant results are more. Also lower values of α requires less magnitude of effect size, which is important with respect to software engineering where collecting large data sets is not always possible.

3. Using multiple accuracy indictors, that measure the same property, might help increase the conclusion validity of an experimental study.

4. Analysis of differences in absolute residuals of competing models is a useful way of comparison, allowing an experimenter to check for model bias as well as to apply significance tests.

5. Use of qualitative criteria based evaluation is a useful way to complement the quantitative evaluation of model outputs.

4.5 Presentation and package

The final step in the process of experimental software engineering is to decide upon the experiment report outline [35]. Wohlin et al [35] provides with a generic structure for such a report, containing an introduction; problem statement; experiment planing; experiment operation; data analysis; data analysis and interpretation. This structure facilitates reporting both general software engineering experiments and also prediction studies in software engineering.

4.5.1 Useful lessons

1. The reporting structure, if followed with all its constituents, supports replication which is one of the important aims of experimental software engineering.

5 Discussion

Evaluation of software engineering prediction systems makes up an important field in experimental software engineering. In part, issues faced by software engineering prediction systems are similar to the broader issues in experimental software engineering. We often encounter noisy and incomplete data, definition of appropriate measures is difficult and there are trade-offs in applying statistical analysis. There are alternatives to select at different stages of the experimentation process when evaluating prediction systems. With so many factors that might influence the credibility of prediction results, it is sensible to make use of best practices and recommendations at various stages of the experimentation process.

We also observe that data set characteristics have a significant impact on getting results with a particular prediction system. Shepperd et al. [30] showed that step-wise regression produced the most accurate predictions for normal and normal+outlier data sets, and machine learning techniques showed better predictive performance on data sets having collinearity, outliers and normal distribution. Also the data splitting scheme into training and test sets and size of training set had significant impact on a model outcome. This supports using cross-validation and preprocessing wherever feasible.

The inherent problems in software engineering data sets have encouraged non-traditional modeling mechanisms but each one of them come with inherent limitations and require further experimentation. Various alternative models offer a choice to the end user in selecting the most appropriate alternative, especially when there is no significant trend in accuracy prediction of a particular model [7].

More and more studies on software fault predictions are making use of an analytical approach that complements the statistical evaluation. There is a realization that "... statistics on its own does not provide scientific explanations" [9].

We believe that it is useful to investigate the modeling of long term behavior, which would validate the true potential of a model on practical grounds. Also, models should ideally be validated to a wide range of commercial software systems (e.g. operating systems, servers, web browsers) [21] as they represent suitable variations in their respective operational profiles. Moreover, there is a need to design new software metrics that incorporate both quantitative and qualitative criteria. Another potential area of future work includes investigating the impacts of cross-validation method chosen on the predictive model's performance.

6 Conclusion

The problems faced by prediction studies in software engineering are not new but interestingly still pose threats to the validity of these studies. One way to move towards better convergence of study results is to follow a process-oriented approach. We found that the basic steps of experimentation in software engineering are relevant to prediction studies; while there are issues specific to each step in the experimentation process which might require more attention with respect to prediction studies.

References

[1] NASA IV&V facility. http://mdp.ivv.nasa.gov/, (10 Oct 08).

[2] Promise repository. http://promisedata.org/, (10 Oct 08).

[3] W. Afzal and R. Torkar. A comparative evaluation of using genetic programming for predicting fault count data. In *Proceedings of the Third International Conference on Software Engineering Advances*. IEEE Computer Society, 2008.

[4] L. Briand, V. Basili, and W. Thomas. A pattern recognition approach for software engineering data analysis. *IEEE Transactions on Software Engineering*, 18(11), 1992.

[5] L. Briand, K. Emam, and S. Morasca. On the application of measurement theory in software engineering. ISERN-95-04.

[6] C. Burgess and M. Lefley. Can genetic programming improve software effort estimation? *Information and Software Technology*, 43(14), 2001.

[7] V. Challagulla, F. Bastani, I.-L. Yen, and R. Paul. Empirical assessment of machine learning based software defect prediction techniques. *10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, 2005.*, 2005.

[8] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7, 2006.

[9] N. Fenton and M. Neil. A critique of software defect prediction models. *IEEE Transactions on Software Engineering*, 25(5), Sep/Oct 1999.

[10] N. Fenton, M. Neil, W. Marsh, P. Hearty, L. Radlinski, and P. Krause. On the effectiveness of early life cycle defect prediction with bayesian nets. *Empirical Software Engineering*, 2008.

[11] N. Fenton and S. Pfleeger. *Sofware Metrics–A Rigorous and Practical Approach, 2nd Edition*. International Thomson Computer Press, Boston, USA, 1996.

[12] A. Ford. *Modeling the environment: an introduction to system dynamic modeling of environmental systems*. Island Press, Washington DC, USA, 1999.

[13] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit. A simulation study of the model evaluation criterion MMRE. *IEEE Transactions on Software Engineering*, 29(11), 2003.

[14] C. Furia, M. Rossi, and D. Mandrioli. Modeling the environment in software-intensive systems. In *Proceedings of the International Workshop on Modeling in Software Engineering*, Washington, USA, 2007. IEEE Computer Society.

[15] K. Gao and T. Khoshgoftaar. A comprehensive empirical study of count models for software fault prediction. *IEEE Transactions on Reliability*, 56(2), June 2007.

[16] A. Gray and S. MacDonell. A comparison of techniques for developing predictive models of software metrics. *Information and Software Technology*, 39(6), 1997.

[17] T. Khoshgoftaar, N. Seliya, and N. Sundaresh. An empirical study of predicting software faults with case-based reasoning. *Software Quality Control*, 14(2), 2006.

[18] B. Kitchenham, H. Al-Khilidar, M. Babar, M. Berry, K. Cox, J. Keung, F. Kurniawati, M. Staples, H. Zhang, and L. Zhu. Evaluating guidelines for empirical software engineering studies. In *ISESE '06: Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, NY, USA, 2006. ACM.

[19] B. Kitchenham, L. Pickard, S. MacDonell, and M. Shepperd. What accuracy statistics really measure. *IEE Proceedings Software*, 148(3), Jun 2001.

[20] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering*, 34(4), 2008.

[21] P. Li, M. Shaw, and J. Herbsleb. Selecting a defect prediction model for maintenance resource planning and software insurance. In *EDSER-5 affiliated with ICSE*, 2003.

[22] M. R. Lyu and A. Nikora. Applying reliability models more effectively. *IEEE Software*, 9(4), 1992.

[23] C. Mair, G. Kadoda, M. Lefley, K. Phalp, C. Schofield, M. Shepperd, and S. Webster. An investigation of machine learning based prediction systems. *Journal of Systems and Software*, 53(1), 2000.

[24] Y. Malaiya, N. Karunanithi, and P. Verma. Predictability measures for software reliability models. *COMPSAC 90*.

[25] I. Myrtveit and E. Stensrud. A controlled experiment to assess the benefits of estimating with analogy and regression models. *IEEE Transactions on Software Engineering*, 25(4), July-Aug. 1999.

[26] I. Myrtveit, E. Stensrud, and M. Shepperd. Reliability and validity in comparative studies of software prediction models. *IEEE Transactions on Software Engineering*, 31(5), May 2005.

[27] A. Nikora and M. Lyu. An experiment in determining software reliability model applicability. *ISSRE*, Oct 1995.

[28] L. Pickard, B. Kitchenham, and S. Linkman. An investigation of analysis techniques for software datasets. In *Proceedings of the 6th International Symposium on Software Metrics*, Washington, DC, USA, 1999. IEEE Computer Society.

[29] M. Shepperd, M. Cartwright, and G. Kadoda. On building prediction systems for software engineers. *Empirical Software Engineering*, 5(3), 2000.

[30] M. Shepperd and G. Kadoda. Comparing software prediction techniques using simulation. *IEEE Transactions on Software Engineering*, 27(11), 2001.

[31] D. Sjoeberg, J. Hannay, O. Hansen, V. Kampenes, A. Karahasanovic, N. Liborg, and A. Rekdal. A survey of controlled experiments in software engineering. *IEEE Transactions on Software Engineering*, 31(9), Sept. 2005.

[32] L. Tian and A. Noore. Computational intelligence methods in software reliability prediction. In *Computational Intelligence in Reliability Engineering*, pages 375–398. Springer Berlin / Heidelberg, 2007.

[33] W. Tichy. Should computer scientists experiment more? *Computer*, 31(5), May 1998.

[34] H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2005.

[35] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers, USA, 2000.